# Derived forms
# Let-binding
# Pairs, tuples, and records

Advanced Compiler Construction and Program Analysis

**Lecture 2**

Innopolis University, Spring 2022

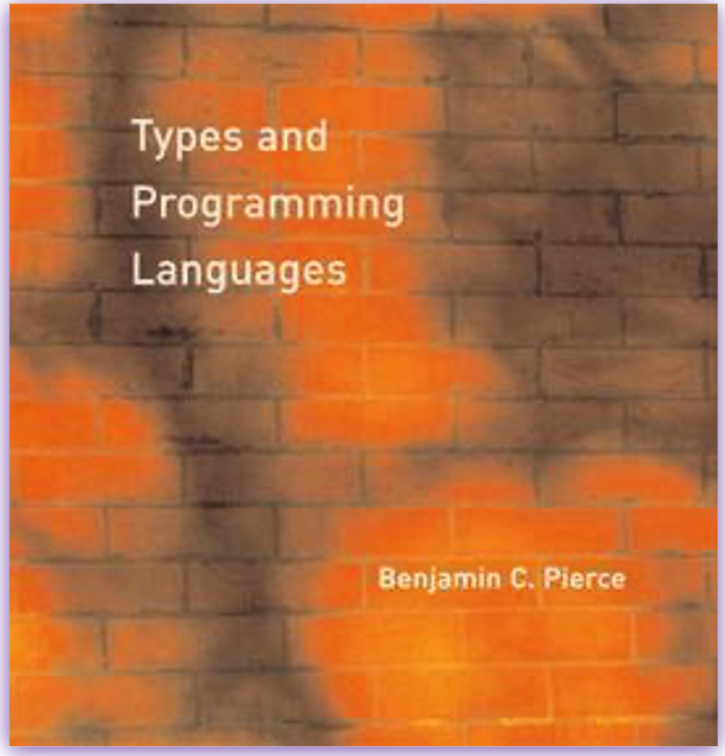
# The topics of this lecture are covered in detail in…

Benjamin C. Pierce.

**Types and Programming Languages**

MIT Press 2002





# Base Types

| | |
|---|---|
| `T ::=` | ***types*** |
| **`Bool`** | *type of booleans* |
| **`Nat`** | *type of natural numbers* |

# Base Types

| T ::= | ***types*** |
| --- | --- |
| **Bool** | *type of booleans* |
| **Nat** | *type of natural numbers* |
| **A** | *(uninterpreted) base type* |

## Base Types

| T ::= | ***types*** |
| --- | --- |
| **Bool** | *type of booleans* |
| **Nat** | *type of natural numbers* |
| **A** | *(uninterpreted) base type* |

$$\lambda x{:}A.x \quad : \quad A \to A$$

This term is completely **valid**, even if there are no known values or evaluation rules for the type **A**.

# Unit Type

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

| | |
|---|---|
| t ::= … | ***terms*** |
| unit | *constant unit* |
| v ::= … | ***values*** |
| **unit** | *unit value* |
| T ::= … | ***types*** |
| **Unit** | *unit type* |

# Sequencing

$$\frac{t_1 \longrightarrow u_1}{t_1;t_2 \longrightarrow u_1;t_2}$$

$$\mathtt{unit};t \longrightarrow t$$

$$\frac{\Gamma \vdash t_1 : \mathtt{Unit} \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1;t_2 : A}$$

| | | |
|---|---|---|
| t ::= … | | ***terms*** |
| | unit | *constant unit* |
| | **t;t** | *sequencing* |
| v ::= … | | ***values*** |
| | **unit** | *unit value* |
| T ::= … | | ***types*** |
| | **Unit** | *unit type* |

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

# Sequencing as a derived form

$$t_1;t_2 := (\lambda x{:}\mathsf{Unit}.t_2)\ t_1$$

| | |
|---|---|
| t ::= … | ***terms*** |
| unit | *constant unit* |
| t;t | *sequencing* |
| v ::= … | ***values*** |
| **unit** | *unit value* |
| T ::= … | ***types*** |
| **Unit** | *unit type* |

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

# Sequencing IS a derived form

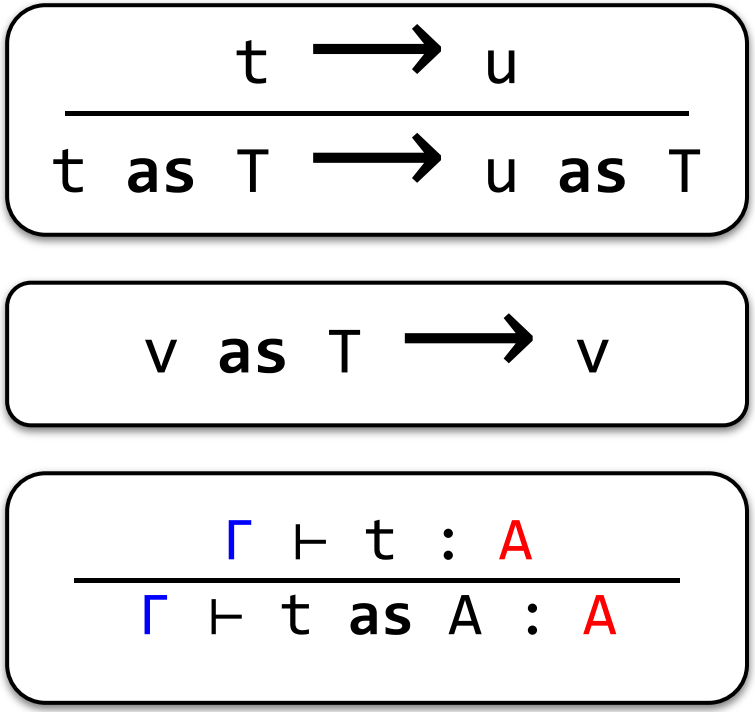
**Theorem 3.1.** Let

1. λS be lambda calculus with Unit type and sequencing,
2. λU be lambda calculus only with Unit type.
3. e : λS → λU be the *elaboration* function, that expands sequences in terms of lambda abstraction

Then

1. `t` ⟶ `u` in λS if and only if `e(t)` ⟶ `e(u)` in λU.
2. `Γ ⊢ t : A` in λS if and only if `Γ ⊢ e(t) : A` in λU.

# Type ascription




```
t ::= …                    terms
    t as T                 type ascription
```

# Type ascription as derived form

**Exercise 3.2.**

1. Show how to formulate type ascription as a derived form. Prove that it is indeed a derived form, meaning that the evaluation and typing rules above correspond to the semantics of derived ascription in a proper way.

2. What changes if we give an “eager” evaluation rule?

# Let binding

$$\begin{array}{l} t ::= \ldots \\ \quad \texttt{let}\ x = t\ \texttt{in}\ t \end{array}$$

***terms***

*let binding*

$$\texttt{let}\ x = v_1\ \texttt{in}\ t_2 \longrightarrow [x \mapsto v_1]t_2$$

$$\frac{t_1 \longrightarrow u_1}{\texttt{let}\ x = t_1\ \texttt{in}\ t_2 \longrightarrow \texttt{let}\ x = u_1\ \texttt{in}\ t_2}$$

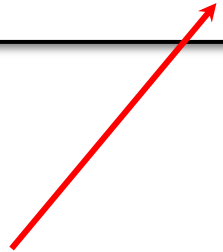
$$\frac{\Gamma \vdash t_1 : A \qquad \Gamma, x : A \vdash t_2 : B}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : B}$$

# Let binding as a derived form

$$\texttt{let } x = t_1 \texttt{ in } t_2 := (\lambda x{:}T.t_2)t_1$$

# Let binding as a derived form

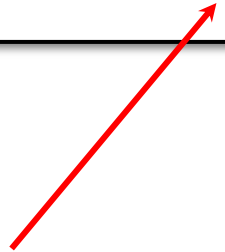
$$\texttt{let } x = t_1 \texttt{ in } t_2 := (\lambda x{:}T.t_2)t_1$$



Where does this **T** come from?

# Let binding as a derived form

$$\texttt{let } x = t_1 \texttt{ in } t_2 := (\lambda x{:}T.t_2)t_1$$



Where does this **T** come from?

It comes from the typing derivation of $t_1$!

# Let binding as a derived form (2)

$$\texttt{let } x = t_1 \texttt{ in } t_2 := [x \mapsto t_1]t_2$$

We could also define let binding using substitution.
Is it a good idea?

# Pairs: syntax

```
t ::= …                    terms
    {t, t}                 pair of terms
    t.1                    first projection
    t.2                    second projection

v ::= …                    values
    {v, v}                 pair of values

T ::= …                    types
      T × T                product type
```

# Pairs: evaluation

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

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

$$\frac{t \longrightarrow u}{t.1 \longrightarrow u.1}$$

$$\frac{t \longrightarrow u}{t.2 \longrightarrow u.2}$$

$$\frac{t_1 \longrightarrow u_1}{\{t_1, t_2\} \longrightarrow \{u_1, t_2\}}$$

$$\frac{t_2 \longrightarrow u_2}{\{v_1, t_2\} \longrightarrow \{v_1, u_2\}}$$

# Pairs: typing

$$\frac{\Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : B}{\Gamma \vdash \{t_1, t_2\} : A \times B}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t.1 : A}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t.2 : B}$$

# Pairs: example

**Exercise 3.3.** Evaluate the following term:

```
{pred 4, if true then false else false}.1
```

# Pairs: example (2)

**Exercise 3.4.** Evaluate the following term:

```
(λx:Nat × Nat. x.2) {pred 4, pred 5}
```

# Tuples: syntax

| | |
|---|---|
| $t ::= \ldots$ | ***terms*** |
| $\{t_i\}$ | *tuple of terms* |
| $t.i$ | *i-th projection* |
| $v ::= \ldots$ | ***values*** |
| $\{v_i\}$ | *tuple of values* |
| $T ::= \ldots$ | ***types*** |
| $\{T_i\}$ | *tuple type* |

# Tuples: evaluation

$$\{v_i\}.k \longrightarrow v_k$$

$$\frac{t \longrightarrow u}{t.k \longrightarrow u.k}$$

$$\frac{t_i \longrightarrow u_i}{\{v_1, \ldots, t_i, \ldots t_n\} \longrightarrow \{v_1, \ldots, u_i, \ldots t_n\}}$$

# Tuples: typing

$$\frac{\textit{for each } i \textit{ from 1 to } n \quad \Gamma \vdash t_i : A_i}{\Gamma \vdash \{t_i\} : \{A_i\}}$$

$$\frac{\Gamma \vdash t : \{A_i\}}{\Gamma \vdash t.k : A_k}$$

# Records: syntax

| | |
|---|---|
| t ::= … | ***terms*** |
| $\{l_i = t_i\}$ | *record* |
| t.l | *projection* |
| | |
| v ::= … | ***values*** |
| $\{l_i = v_i\}$ | *record value* |
| | |
| T ::= … | ***types*** |
| $\{l_i : T_i\}$ | *record type* |

# Records: evaluation

$$\{\ldots,\ l = v,\ \ldots\}.l \longrightarrow v$$

$$\frac{t \longrightarrow u}{t.l \longrightarrow u.l}$$

$$\frac{t \longrightarrow u}{\{l_1 = v_1,\ \ldots,\ l_i = t_i,\ \ldots,\ l_n = t_n\} \longrightarrow \{l_1 = v_1,\ \ldots,\ l_i = u_i,\ \ldots,\ l_n = t_n\}}$$

# Records: typing

$$\frac{\textit{for each } i \textit{ from 1 to } n \quad \Gamma \vdash t_i : A_i}{\Gamma \vdash \{l_i = t_i\} : \{l_i : A_i\}}$$

$$\frac{\Gamma \vdash t : \{l_i : A_i\}}{\Gamma \vdash t.l_k : A_k}$$

# Equivalence of record values and types

Are the following values the same?

```
{a = pred 4, b = pred 5}
{b = pred 5, a = pred 4}
```

# Equivalence of record values and types

Are the following values the same?

```
{a = pred 4, b = pred 5}
{b = pred 5, a = pred 4}
```

Are these types the same?

```
{a : Nat, b : Bool}
{b : Bool, a : Nat}
```

# Pattern matching records: syntax

```
p ::=                          patterns
   x                           variable
   {l_i = p_i}                 record pattern

t ::= …                        terms
   let p = t in t              let binding
```

# Pattern matching records: evaluation

$$\texttt{let p = } v_1 \texttt{ in } t_2 \longrightarrow \texttt{match(p, } v_1\texttt{)} t_2$$

$$\frac{t_1 \longrightarrow u_1}{\texttt{let p = } t_1 \texttt{ in } t_2 \longrightarrow \texttt{let p = } u_1 \texttt{ in } t_2}$$

# Pattern matching records: matching

$$\mathbf{match}(x, v) := [x \mapsto v]$$

$$\mathbf{match}(\{l_i = p_i\}, \{l_i = v_i\}) := \mathrm{match}(p_1, v_1) \ldots \mathrm{match}(p_i, v_i) \ldots \mathrm{match}(p_n, v_n)$$

# Summary

- ❏ Base Types
- ❏ Derived forms
- ❏ Pairs, tuples, records

## Summary

- ❏ Base Types
- ❏ Derived forms
- ❏ Pairs, tuples, records

# See you next time!