Simply typed λ-calculus

Advanced Compiler Construction and Program Analysis

Lecture 2

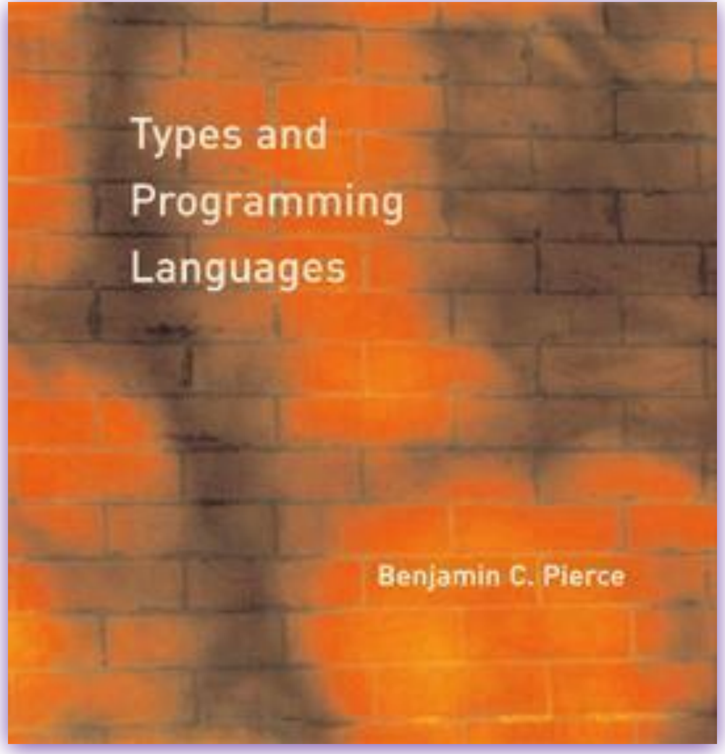
Innopolis University, Spring 2022

The topics of this lecture are covered in detail in...

Benjamin C. Pierce.

Types and Programming Languages

MIT Press 2002



Untyped Arithmetic Expressions. Syntax

`t ::=`	***terms***
`true`	*constant true*
`false`	*constant false*
`if t then t else t`	*conditional*
`0`	*constant zero*
`succ t`	*successor*
`pred t`	*predecessor*
`iszero t`	*zero test*

Untyped Arithmetic Expressions. Syntax

Recall, that any term **`t`** can

1. Either evaluate to a ***value***, or
2. Get ***stuck***.

`v ::=`	***values***
`true`	*true value*
`false`	*false value*
`nv`	*numeric value*
`nv ::=`	***numeric values***
`0`	*zero value*
`succ` `nv`	*successor value*

Typing relation. Intuition

Recall, that any term `t` can

1. Either evaluate to a ***value***, or
2. Get ***stuck***.

We say that *"a term `t` has type **T**"* to mean that t "obviously" evaluates to a value corresponding to type **T**.

```
v ::=                 values
    true              true value
    false             false value
    nv                numeric value

nv ::=                numeric values
    0                 zero value
    succ nv           successor value
```

Typing relation. Static and conservative

if true **then** false **else** true *has type* **Bool**

Typing relation. Static and conservative

if true **then** false **else** true *has type* **Bool**

pred (succ (pred (succ 0))) *has type* **Nat**

Typing relation. Static and conservative

if true **then** false **else** true *has type* **Bool**

pred (succ (pred (succ 0))) *has type* **Nat**

if (iszero 0) **then** 0 **else** false ***is ill-typed***

Typing relation. Static and conservative

if true **then** false **else** true *has type* **Bool**
pred (succ (pred (succ 0))) *has type* **Nat**
if (iszero 0) **then** 0 **else** false *is ill-typed*
if true **then** 0 **else** false *is ill-typed*

Boolean Expressions. Typing relation

`t : T`

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

Boolean Expressions. Typing relation

`t : T`

```
T ::=
    Bool
```

types

type of booleans

`true : Bool`

`false : Bool`

Boolean Expressions. Typing relation

$$\texttt{t : T}$$

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

$$\texttt{true : Bool}$$

$$\texttt{false : Bool}$$

$$\frac{t_1 : \texttt{Bool} \qquad t_2 : T \qquad t_3 : T}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T}$$

Numerical Expressions. Typing

`t : T`

```
T ::=
...
    Nat
```

types

type of natural numbers

Numerical Expressions. Typing

```
t : T
```

```
T ::=
...
    Nat
```

types

type of natural numbers

```
0 : Nat
```

Numerical Expressions. Typing

`t : T`

`T ::=` ***types***

`...` *type of natural numbers*

`Nat`

$$\mathtt{0 : Nat}$$

$$\frac{\mathtt{t : Nat}}{\mathtt{succ\ t : Nat}}$$

Numerical Expressions. Typing

$t : T$

T ::= ***types***

... *type of natural numbers*

Nat

$$0 : \mathrm{Nat}$$

$$\frac{t : \mathrm{Nat}}{\mathrm{succ}\ t : \mathrm{Nat}}$$

$$\frac{t : \mathrm{Nat}}{\mathrm{pred}\ t : \mathrm{Nat}}$$

$$\frac{t : \mathrm{Nat}}{\mathrm{iszero}\ t : \mathrm{Bool}}$$

Expressions. Typing example

$$\frac{\dots}{\texttt{pred (\textbf{if} iszero 0 \textbf{then} succ 0 \textbf{else} 0) : \textbf{Nat}}}$$

Expressions. Typing example

$$\dfrac{\dfrac{\dots}{\textbf{if}\ \texttt{iszero 0}\ \textbf{then}\ \texttt{succ 0}\ \textbf{else}\ \texttt{0} : \textbf{Nat}}}{\texttt{pred}\ \textbf{(}\textbf{if}\ \texttt{iszero 0}\ \textbf{then}\ \texttt{succ 0}\ \textbf{else}\ \texttt{0)} : \textbf{Nat}}$$

Expressions. Typing example

$$\dfrac{\dfrac{\dfrac{\cdots}{\texttt{iszero 0} : \mathbf{Bool}} \qquad \dfrac{\cdots}{\texttt{succ 0} : \mathbf{Nat}} \qquad \texttt{0} : \mathbf{Nat}}{\mathbf{if}\ \texttt{iszero 0}\ \mathbf{then}\ \texttt{succ 0}\ \mathbf{else}\ \texttt{0} : \mathbf{Nat}}}{\texttt{pred}\ \mathbf{(if}\ \texttt{iszero 0}\ \mathbf{then}\ \texttt{succ 0}\ \mathbf{else}\ \texttt{0)} : \mathbf{Nat}}$$

Expressions. Typing example

$$
\dfrac{\dfrac{\dfrac{0 : \mathbf{Nat}}{\texttt{iszero}\ 0 : \mathbf{Bool}} \qquad \dfrac{0 : \mathbf{Nat}}{\texttt{succ}\ 0 : \mathbf{Nat}} \qquad 0 : \mathbf{Nat}}{\mathbf{if}\ \texttt{iszero}\ 0\ \mathbf{then}\ \texttt{succ}\ 0\ \mathbf{else}\ 0 : \mathbf{Nat}}}{\texttt{pred}\ (\mathbf{if}\ \texttt{iszero}\ 0\ \mathbf{then}\ \texttt{succ}\ 0\ \mathbf{else}\ 0) : \mathbf{Nat}}
$$

Expressions. Typing example

$$\dfrac{\dfrac{\dfrac{\texttt{0 : }\textbf{\texttt{Nat}}}{\texttt{iszero 0 : }\textbf{\texttt{Bool}}} \qquad \dfrac{\texttt{0 : }\textbf{\texttt{Nat}}}{\texttt{succ 0 : }\textbf{\texttt{Nat}}} \qquad \texttt{0 : }\textbf{\texttt{Nat}}}{\textbf{\texttt{if}}\texttt{ iszero 0 }\textbf{\texttt{then}}\texttt{ succ 0 }\textbf{\texttt{else}}\texttt{ 0 : }\textbf{\texttt{Nat}}}}{\texttt{pred (}\textbf{\texttt{if}}\texttt{ iszero 0 }\textbf{\texttt{then}}\texttt{ succ 0 }\textbf{\texttt{else}}\texttt{ 0) : }\textbf{\texttt{Nat}}}$$

If, for a given term **`t`**, there exists a type **`T`**, such that **`t : T`**, then we say that **`t`** is ***typeable*** (or that **`t`** is ***well-typed***). Otherwise, we say that **`t`** is ***ill-typed***.

Typing lambda abstraction

❖ What is the type of a lambda abstraction **`λx.t`**?

Typing lambda abstraction

- What is the type of a lambda abstraction **`λx.t`**?

- It is a function, but simply introduction type **`Function`** is not enough: **`λx.succ x`** and **`λx.λy.iszero x`** are both functions, but with different number of arguments and result types.

Typing lambda abstraction

- What is the type of a lambda abstraction **`λx.t`**?

- It is a function, but simply introduction type **`Function`** is not enough: **`λx.succ x`** and **`λx.λy.iszero x`** are both functions, but with different number of arguments and result types.

- Instead, we want types like $\mathtt{T_1} \to \mathtt{T_2}$, specifying input and output types.

Typing lambda abstraction

- What is the type of a lambda abstraction **`λx.t`**?
- It is a function, but simply introduction type **`Function`** is not enough: **`λx.succ x`** and **`λx.λy.iszero x`** are both functions, but with different number of arguments and result types.
- Instead, we want types like $\mathtt{T_1} \to \mathtt{T_2}$, specifying input and output types.
- But what is the type of the argument of **`λx.0`**?

Typing lambda abstraction

- What is the type of a lambda abstraction **`λx.t`**?
- It is a function, but simply introduction type **`Function`** is not enough: **`λx.succ x`** and **`λx.λy.iszero x`** are both functions, but with different number of arguments and result types.
- Instead, we want types like $\mathtt{T_1} \rightarrow \mathtt{T_2}$, specifying input and output types.
- But what is the type of the argument of **`λx.0`** ?
- To aid type checking, we ***explicitly*** type the argument: **`λx:T.t`**

Simply Typed λ-calculus. Typing

$\Gamma \vdash t : T$

```
t ::= ...
    x
    λx:T.t
    t t
```

terms
variable
abstraction
application

```
T ::= ...
   T → T
```

types
function type

Simply Typed λ-calculus. Typing

$\Gamma \vdash t : T$

```
t ::= ...             terms
    x                 variable
    λx:T.t            abstraction
    t t               application

T ::= ...             types
   T → T              function type
```

$\Gamma, x{:}T \vdash x : T$

Simply Typed λ-calculus. Typing

$$\Gamma \vdash t : T$$

```
t ::= ...          terms
    x              variable
    λx:T.t         abstraction
    t t            application

T ::= ...          types
   T → T           function type
```

$$\Gamma, x{:}T \vdash x : T$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x{:}A.\ t) : A \to B}$$

Simply Typed λ-calculus. Typing

$$\Gamma \vdash t : T$$

`t ::= ...`	***terms***
`x`	*variable*
`λx:T.t`	*abstraction*
`t t`	*application*
`T ::= ...`	***types***
`T → T`	*function type*

$$\Gamma, x{:}T \vdash x : T$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x{:}A.\ t) : A \to B}$$

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

Simply Typed λ-calculus. Typing example

$$\frac{\dots}{\vdash \lambda x{:}\mathbf{Nat}.\ \lambda y{:}\mathbf{Nat}.\ \mathbf{if}\ \mathtt{iszero}\ x\ \mathbf{then}\ y\ \mathbf{else}\ \mathtt{succ}\ y \;:\; \mathbf{Nat} \to (\mathbf{Nat} \to \mathbf{Nat})}$$

Simply Typed λ-calculus. Typing example

$$\dfrac{\dfrac{\ldots}{x:\mathtt{Nat} \vdash \lambda y:\mathtt{Nat}.\ \mathbf{if}\ \mathtt{iszero}\ x\ \mathbf{then}\ y\ \mathbf{else}\ \mathtt{succ}\ y : \mathtt{Nat} \to \mathtt{Nat}}}{\vdash \lambda x:\mathtt{Nat}.\ \lambda y:\mathtt{Nat}.\ \mathbf{if}\ \mathtt{iszero}\ x\ \mathbf{then}\ y\ \mathbf{else}\ \mathtt{succ}\ y : \mathtt{Nat} \to (\mathtt{Nat} \to \mathtt{Nat})}$$

Simply Typed λ-calculus. Typing example

$$\dfrac{\dfrac{\dfrac{\ldots}{\texttt{x:Nat,y:Nat} \vdash \texttt{if iszero x then y else succ y} : \texttt{Nat}}}{\texttt{x:Nat} \vdash \lambda\texttt{y:Nat. if iszero x then y else succ y} : \texttt{Nat} \to \texttt{Nat}}}{\vdash \lambda\texttt{x:Nat.}\ \lambda\texttt{y:Nat. if iszero x then y else succ y} : \texttt{Nat} \to (\texttt{Nat} \to \texttt{Nat})}$$

Simply Typed λ-calculus. Typing example

$$
\dfrac{\dfrac{\dots}{x{:}\mathbf{Nat},y{:}\mathbf{Nat} \vdash \textbf{if}\ \texttt{iszero x}\ \textbf{then}\ \texttt{y}\ \textbf{else}\ \texttt{succ y} : \mathbf{Nat}}}{\dfrac{x{:}\mathbf{Nat} \vdash \lambda y{:}\mathbf{Nat}.\ \textbf{if}\ \texttt{iszero x}\ \textbf{then}\ \texttt{y}\ \textbf{else}\ \texttt{succ y} : \mathbf{Nat} \to \mathbf{Nat}}{\vdash \lambda x{:}\mathbf{Nat}.\ \lambda y{:}\mathbf{Nat}.\ \textbf{if}\ \texttt{iszero x}\ \textbf{then}\ \texttt{y}\ \textbf{else}\ \texttt{succ y} : \mathbf{Nat} \to (\mathbf{Nat} \to \mathbf{Nat})}}
$$

$\Gamma := x{:}\mathbf{Nat},y{:}\mathbf{Nat}$

Simply Typed λ-calculus. Typing example

$$\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{\Gamma \vdash \mathtt{iszero}\ \mathtt{x} : \mathbf{Bool}} \qquad \dfrac{}{\Gamma \vdash \mathtt{y} : \mathbf{Nat}} \qquad \dfrac{\cdots}{\Gamma \vdash \mathtt{succ}\ \mathtt{y} : \mathbf{Nat}}}{\mathtt{x}{:}\mathbf{Nat},\mathtt{y}{:}\mathbf{Nat} \vdash \mathbf{if}\ \mathtt{iszero}\ \mathtt{x}\ \mathbf{then}\ \mathtt{y}\ \mathbf{else}\ \mathtt{succ}\ \mathtt{y} : \mathbf{Nat}}}{\mathtt{x}{:}\mathbf{Nat} \vdash \lambda \mathtt{y}{:}\mathbf{Nat}.\ \mathbf{if}\ \mathtt{iszero}\ \mathtt{x}\ \mathbf{then}\ \mathtt{y}\ \mathbf{else}\ \mathtt{succ}\ \mathtt{y} : \mathbf{Nat} \to \mathbf{Nat}}}{\vdash \lambda \mathtt{x}{:}\mathbf{Nat}.\ \lambda \mathtt{y}{:}\mathbf{Nat}.\ \mathbf{if}\ \mathtt{iszero}\ \mathtt{x}\ \mathbf{then}\ \mathtt{y}\ \mathbf{else}\ \mathtt{succ}\ \mathtt{y} : \mathbf{Nat} \to (\mathbf{Nat} \to \mathbf{Nat})}$$

$$\Gamma := \mathtt{x}{:}\mathbf{Nat},\mathtt{y}{:}\mathbf{Nat}$$

Simply Typed λ-calculus. Typing example

$$
\dfrac{
  \dfrac{\dfrac{}{\Gamma \vdash x : \mathtt{Nat}}}{\Gamma \vdash \mathtt{iszero}\ x : \mathtt{Bool}}
  \qquad
  \dfrac{}{\Gamma \vdash y : \mathtt{Nat}}
  \qquad
  \dfrac{\dfrac{}{\Gamma \vdash y : \mathtt{Nat}}}{\Gamma \vdash \mathtt{succ}\ y : \mathtt{Nat}}
}{
  \dfrac{
    x{:}\mathtt{Nat}, y{:}\mathtt{Nat} \vdash \mathbf{if}\ \mathtt{iszero}\ x\ \mathbf{then}\ y\ \mathbf{else}\ \mathtt{succ}\ y : \mathtt{Nat}
  }{
    \dfrac{
      x{:}\mathtt{Nat} \vdash \lambda y{:}\mathtt{Nat}.\ \mathbf{if}\ \mathtt{iszero}\ x\ \mathbf{then}\ y\ \mathbf{else}\ \mathtt{succ}\ y : \mathtt{Nat} \to \mathtt{Nat}
    }{
      \vdash \lambda x{:}\mathtt{Nat}.\ \lambda y{:}\mathtt{Nat}.\ \mathbf{if}\ \mathtt{iszero}\ x\ \mathbf{then}\ y\ \mathbf{else}\ \mathtt{succ}\ y : \mathtt{Nat} \to (\mathtt{Nat} \to \mathtt{Nat})
    }
  }
}
$$

$\Gamma := x{:}\mathtt{Nat}, y{:}\mathtt{Nat}$

Attendance

https://baam.duckdns.org

Type Safety (a.k.a. Soundness of a Type System)

Idea is that well-typed terms do not “go wrong”.
“Go wrong” = evaluation is stuck

Progress: a well-typed term is not stuck — it is either a value or it can be reduced to another term.

Preservation: if a well-typed term can be reduced to some term `t`, then `t` is also well-typed.

Safety = Progress + Preservation

Properties of Typing: Inversion

Lemma 2.1.

1. If $\Gamma \vdash \texttt{true} : R$, then $R = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : R$, then $R = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$, then $\Gamma \vdash t_1 : \texttt{Bool}$, and $\Gamma \vdash t_2 : R$, and $\Gamma \vdash t_3 : R$.
4. If $\Gamma \vdash x : R$, then $(x : R) \in \Gamma$.
5. If $\Gamma \vdash (t_1\ t_2) : R$, then there exists type T, such that $\Gamma \vdash t_1 : T \to R$ and $\Gamma \vdash t_2 : T$.
6. If $\Gamma \vdash (\lambda x{:}T_1.\ t) : R$, then $R = T_1 \to R_2$, for some R_2, such that $\Gamma, x : T_1 \vdash t : R_2$.

Properties of Typing: Inversion

Lemma 2.1.

1. If $\Gamma \vdash \texttt{true} : R$, then $R = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : R$, then $R = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$, then $\Gamma \vdash t_1 : \texttt{Bool}$, and $\Gamma \vdash t_2 : R$, and $\Gamma \vdash t_3 : R$.
4. If $\Gamma \vdash x : R$, then $(x : R) \in \Gamma$.
5. If $\Gamma \vdash (t_1\ t_2) : R$, then there exists type T, such that $\Gamma \vdash t_1 : T \to R$ and $\Gamma \vdash t_2 : T$.
6. If $\Gamma \vdash (\lambda x{:}T_1.\ t) : R$, then $R = T_1 \to R_2$, for some R_2, such that $\Gamma, x : T_1 \vdash t : R_2$.

Properties of Typing: Inversion

Lemma 2.1.

1. If $\Gamma \vdash \texttt{true} : R$, then $R = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : R$, then $R = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$,
   then $\Gamma \vdash t_1 : \texttt{Bool}$, and $\Gamma \vdash t_2 : R$, and $\Gamma \vdash t_3 : R$.
4. If $\Gamma \vdash x : R$, then $(x : R) \in \Gamma$.
5. If $\Gamma \vdash (t_1\ t_2) : R$, then there exists type T,
   such that $\Gamma \vdash t_1 : T \to R$ and $\Gamma \vdash t_2 : T$.
6. If $\Gamma \vdash (\lambda x{:}T_1.\ t) : R$, then $R = T_1 \to R_2$,
   for some R_2, such that $\Gamma, x : T_1 \vdash t : R_2$.

Properties of Typing: Inversion

Lemma 2.1.

1. If $\Gamma \vdash \mathtt{true} : R$, then $R = \mathtt{Bool}$.
2. If $\Gamma \vdash \mathtt{false} : R$, then $R = \mathtt{Bool}$.
3. If $\Gamma \vdash \mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : R$, then $\Gamma \vdash t_1 : \mathtt{Bool}$, and $\Gamma \vdash t_2 : R$, and $\Gamma \vdash t_3 : R$.
4. If $\Gamma \vdash x : R$, then $(x : R) \in \Gamma$.
5. If $\Gamma \vdash (t_1\ t_2) : R$, then there exists type T, such that $\Gamma \vdash t_1 : T \to R$ and $\Gamma \vdash t_2 : T$.
6. If $\Gamma \vdash (\lambda x{:}T_1.\ t) : R$, then $R = T_1 \to R_2$, for some R_2, such that $\Gamma, x : T_1 \vdash t : R_2$.

Properties of Typing: Inversion

Lemma 2.1.

1. If $\Gamma \vdash \mathtt{true} : \mathrm{R}$, then $\mathrm{R} = \mathtt{Bool}$.
2. If $\Gamma \vdash \mathtt{false} : \mathrm{R}$, then $\mathrm{R} = \mathtt{Bool}$.
3. If $\Gamma \vdash \mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : \mathrm{R}$, then $\Gamma \vdash t_1 : \mathtt{Bool}$, and $\Gamma \vdash t_2 : \mathrm{R}$, and $\Gamma \vdash t_3 : \mathrm{R}$.
4. If $\Gamma \vdash x : \mathrm{R}$, then $(x : \mathrm{R}) \in \Gamma$.
5. If $\Gamma \vdash (t_1\ t_2) : \mathrm{R}$, then there exists type T, such that $\Gamma \vdash t_1 : T \to R$ and $\Gamma \vdash t_2 : T$.
6. If $\Gamma \vdash (\lambda x{:}T_1.\ t) : \mathrm{R}$, then $\mathrm{R} = T_1 \to R_2$, for some R_2, such that $\Gamma, x : T_1 \vdash t : R_2$.

Properties of Typing: Inversion

Lemma 2.1.

1. If $\Gamma \vdash \texttt{true} : R$, then $R = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : R$, then $R = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$, then $\Gamma \vdash t_1 : \texttt{Bool}$, and $\Gamma \vdash t_2 : R$, and $\Gamma \vdash t_3 : R$.
4. If $\Gamma \vdash x : R$, then $(x : R) \in \Gamma$.
5. If $\Gamma \vdash (t_1\ t_2) : R$, then there exists type T, such that $\Gamma \vdash t_1 : T \to R$ and $\Gamma \vdash t_2 : T$.
6. If $\Gamma \vdash (\lambda x{:}T_1.\ t) : R$, then $R = T_1 \to R_2$, for some R_2, such that $\Gamma, x : T_1 \vdash t : R_2$.

Uniqueness of Types (simply typed λ-calculus)

Theorem 2.2. Each term `t` has at most one type.
That is, if `t` is typeable, then its type is unique.
Moreover, there is just one derivation of this typing build from the typing rules for boolean and numeric expressions.

Proof. Straightforward by structural induction on `t`, using **Lemma 2.1** appropriately for each case.

Properties of Typing: Canonical forms

Lemma 2.3.

1. If $\Gamma \vdash$ `v : Bool` and v is a value,
   then `v = false` or `v = true`.

2. If $\Gamma \vdash$ `v : Nat` and v is a value,
   then `v = 0` or `v = succ nv`.

3. If $\Gamma \vdash$ `v : A → B` and v is a value,
   then `v = λx:A.t`.

Properties of Typing: Progress

Theorem 2.4. Suppose $\Gamma \vdash t : T$ then either

1. t is a ***value***, or
2. there exists t', such that $t \longrightarrow t'$

Properties of Typing: Preservation

Theorem 2.5.

Suppose $\Gamma \vdash \texttt{t} : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t'}$, then $\Gamma \vdash \texttt{t'} : \texttt{T}$.

Properties of Typing: Preservation

Theorem 2.5.

Suppose $\Gamma \vdash \texttt{t} : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t'}$, then $\Gamma \vdash \texttt{t'} : \texttt{T}$.

Exercise 2.6. Is the following proposition TRUE or FALSE?
For all terms $\texttt{t}$ and $\texttt{t'}$,

if $\Gamma \vdash \texttt{t'} : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t'}$, then $\Gamma \vdash \texttt{t} : \texttt{T}$.

Properties of Typing: Permutations

Lemma 2.7.
Suppose $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and Δ is a permutation of Γ,
then $\Delta \vdash \mathtt{t} : \mathtt{T}$.

Proof. Straightforward by induction on typing contexts.

Properties of Typing: Weakening

Lemma 2.8.
Suppose $\Gamma \vdash t : T$, then $\Gamma, x{:}S \vdash t : T$.

Proof. Straightforward by induction on typing contexts.

Properties of Typing: Weakening

Lemma 2.8.
Suppose $\Gamma \vdash \texttt{t : T}$, then $\Gamma, \texttt{x:S} \vdash \texttt{t : T}$.

Proof. Straightforward by induction on typing contexts.

Remark. Some type systems (e.g. linear type systems) do not have weakening (among other properties), thus making sure that every variable in the typing context is actually used (occurs) in the typed term.

Curry-Howard Correspondence

type **A**		proposition **A**
$\mathbf{A} \to \mathbf{B}$		$\mathbf{A} \Rightarrow \mathbf{B}$
$\mathbf{A} \times \mathbf{B}$		**A and B**
program **t** of type **A**	*correspond to*	proof **t** of prop. **A**
type **A** is inhabitable		prop. **A** is provable
program evaluation		proof simplification

Type Erasure

Definition 2.9.

```
erase(x)       = x
erase(λx:T.t) = λx.erase(t)
erase(t₁ t₂)  = erase(t₁) erase(t₂)
```

Type Erasure

Definition 2.9.

```
erase(x)         = x
erase(λx:T.t) = λx.erase(t)
erase(t₁ t₂)   = erase(t₁) erase(t₂)
```

Theorem 2.10.

1. If $\mathtt{t} \longrightarrow \mathtt{t'}$, then $\mathtt{erase(t)} \longrightarrow \mathtt{erase(t')}$.
2. If $\mathtt{erase(t)} \longrightarrow \mathtt{m}$, then there exists some term $\mathtt{t'}$, such that $\mathtt{t} \longrightarrow \mathtt{t'}$ and $\mathtt{erase(t')} = \mathtt{m}$.

Summary

- Typing relation and Typing context
- Properties of Typing relation
- Simply Typed λ-calculus

Summary

- ❏ Typing relation and Typing context
- ❏ Properties of Typing relation
- ❏ Simply Typed λ-calculus

See you next time!