CENG 2010 - Programming Language Concepts

Weeks 10-11: Simply Typed λ-Calculus ($\lambda^{\rightarrow}$)

Burak Ekici

May 15 – May 22, 2023

Outline

1. Typing In General
2. STLC($\lambda^{\rightarrow}$)

Need for Types

- In (untyped) λ-Calculus, we can easily misuse terms:

$$\begin{aligned} \text{false} &:= \lambda x.\lambda y.y \\ 0 &:= \lambda s.\lambda z.z \end{aligned}$$

Need for Types

- In (untyped) λ-Calculus, we can easily misuse terms:

$$\begin{array}{lll} \text{false} & := & \lambda x.\lambda y.y \\ 0 & := & \lambda s.\lambda z.z \\ \text{false } 0 & =_\beta & \lambda y.y \end{array}$$

Need for Types

- In (untyped) λ-Calculus, we can easily misuse terms:

$$\begin{array}{lll} \text{false} & := & \lambda x.\lambda y.y \\ 0 & := & \lambda s.\lambda z.z \\ \text{false } 0 & =_\beta & \lambda y.y \end{array}$$

- there is an obvious need to guarantee that function parameters are “valid” in terms of function application to prevent errors during the evaluation

Need for Types

- In (untyped) λ-Calculus, we can easily misuse terms:

$$\begin{array}{lcl} \text{false} & := & \lambda x.\lambda y.y \\ 0 & := & \lambda s.\lambda z.z \\ \text{false } 0 & =_\beta & \lambda y.y \end{array}$$

- there is an obvious need to guarantee that function parameters are “valid” in terms of function application to prevent errors during the evaluation
- this is in fact the fundamental purpose of type systems

Type Systems in General

- type system is a set of rules that assigns a property called a type to the terms (perhaps to other various constructs) in a program with a purpose to reduce possibilities for bugs, and evaluation errors

Type Systems in General

- type system is a set of rules that assigns a property called a type to the terms (perhaps to other various constructs) in a program with a purpose to reduce possibilities for bugs, and evaluation errors
- some mechanism to distinguish "good" and "bad" programs

0 + 1	is well-typed	good
0 + false	is ill-typed	bad
if false then 10 else 20	is well-typed	good
1 + (if true then 10 else false)	is ill-typed	bad

Type Systems in General (cont'd)

- main point is to classify terms into types
- given a set of (inductively generated) types

$$Ty \quad := \quad T_1 \mid T_2 \mid T_3 \mid \ldots$$

- a term t might be of type $T_1, T_2, T_3, \ldots$

Type Systems in General (cont'd)

- main point is to classify terms into types
- given a set of (inductively generated) types

$$Ty \quad := \quad T_1 \mid T_2 \mid T_3 \mid \ldots$$

- a term t might be of type $T_1, T_2, T_3, \ldots$ (thanks to a typing relation)

Typing Relations in General

- typing relation ":" assigns types to terms

Typing Relations in General

- typing relation ":" assigns types to terms
- formally, a typing relation is a partial binary predicate ":" $: \mathcal{E} \times \mathcal{T}y \rightarrow Bool$ where
    - $\mathcal{E}$ is the set of all possible terms
    - $\mathcal{T}y$ is the set of all possible types

Typing Relations in General

- typing relation ":" assigns types to terms
- formally, a typing relation is a partial binary predicate ":" $: \mathcal{E} \times \mathcal{T}y \rightarrow \mathit{Bool}$ where
  - $\mathcal{E}$ is the set of all possible terms
  - $\mathcal{T}y$ is the set of all possible types
- some related notions:

$$\begin{array}{lcl} \text{language} & := & \text{as a set } \mathcal{E} \text{ of all possible terms} \\ \text{type language} & := & \text{as a set } \mathcal{T}y \text{ of all possible types} \\ \text{typing relation} & := & \text{as a partial relation ``:''} \subseteq \mathcal{E} \times \mathcal{T}y \end{array}$$

Typing Relations in General

- typing relation ":" assigns types to terms
- formally, a typing relation is a partial binary predicate ":" $: \mathcal{E} \times \mathcal{T}y \to \mathit{Bool}$ where
  - $\mathcal{E}$ is the set of all possible terms
  - $\mathcal{T}y$ is the set of all possible types
- some related notions:

language	:=	as a set $\mathcal{E}$ of all possible terms
type language	:=	as a set $\mathcal{T}y$ of all possible types
typing relation	:=	as a partial relation ":" $\subseteq \mathcal{E} \times \mathcal{T}y$

- categorical approach is slightly different:

language	:=	internal language of a certain category $\mathcal{C}$
types	:=	objects of $\mathcal{C}$
terms	:=	arrows of $\mathcal{C}$

Outline

1. Typing In General
2. STLC($\lambda^{\rightarrow}$)

Definition (Simply Typed λ-Calculus ($\lambda^{\rightarrow}$))

$$
\begin{array}{lll}
\text{Types } A, B, C, \ldots := & & \\
\quad \mid & G, G', G'', \ldots & \text{“ground” types} \\
\quad \mid & \mathsf{unit} & \text{unit type} \\
\quad \mid & A \times B & \text{product type} \\
\quad \mid & A \rightarrow B & \text{function type}
\end{array}
$$

Definition (Simply Typed λ-Calculus ($\lambda^{\rightarrow}$))

Types $A, B, C, \ldots :=$

$\mid$	$G, G', G'', \ldots$	"ground" types
$\mid$	unit	unit type
$\mid$	$A \times B$	product type
$\mid$	$A \to B$	function type

Terms $s, t, r :=$

$\mid$	c^A	constants (of given type A)
$\mid$	x	variable (countable many)
$\mid$	$()$	unit value
$\mid$	(s, t)	pair
$\mid$	$\mathsf{fst}\ t$	first pair projection
$\mid$	$\mathsf{snd}\ t$	second pair projection
$\mid$	$\lambda x\colon A.\, t$	function abstraction
$\mid$	$s\ t$	function application

Example (term examples)

- $\lambda z : (A \to B) \times (A \to C).\, \lambda x : A.\, ((\mathrm{fst}\ z)\ x, (\mathrm{snd}\ z)\ x)$

Example (term examples)

- $\lambda z\colon (A \to B) \times (A \to C).\, \lambda x\colon A.\, ((\mathrm{fst}\ z)\ x, (\mathrm{snd}\ z)\ x)$
- $\lambda z\colon A \to (B \times C).\, (\lambda x\colon A.\, \mathrm{fst}\ (z\ x), \lambda y\colon A.\, \mathrm{snd}\ (z\ y))$

Example (term examples)

- $\lambda z \colon (A \to B) \times (A \to C).\, \lambda x \colon A.\, ((\mathsf{fst}\ z)\ x, (\mathsf{snd}\ z)\ x)$
- $\lambda z \colon A \to (B \times C).\, (\lambda x \colon A.\, \mathsf{fst}\ (z\ x), \lambda y \colon A.\, \mathsf{snd}\ (z\ y))$
- $\lambda z \colon A \to (B \times C).\, \lambda x \colon A.\, ((\mathsf{fst}\ z)\ x, (\mathsf{snd}\ z)\ x)$

Definition ($\lambda^{\rightarrow}$ typing relation: $\Gamma \vdash t : A$)

Γ ranges over typing environments (or typing contexts)

$$
\begin{array}{lll}
\Gamma := & & \\
\quad | & [\,] & \text{“empty” environment} \\
\quad | & \Gamma, x : A & \text{“non-empty” environment}
\end{array}
$$

Definition ($\lambda^{\rightarrow}$ typing relation: $\Gamma \vdash t : A$)

Γ ranges over typing environments (or typing contexts)

$$
\begin{array}{lll}
\Gamma := & & \\
\quad \mid & [\,] & \text{“empty” environment} \\
\quad \mid & \Gamma, x : A & \text{“non-empty” environment}
\end{array}
$$

typing environments are comma-separated snoc-lists of (variable,type)-pairs – in fact only the lists whose variables are mutually distinct get used

Definition ($\lambda^{\rightarrow}$ typing relation: $\Gamma \vdash t : A$)

Γ ranges over typing environments (or typing contexts)

$$\begin{array}{lll} \Gamma := & & \\ \mid & [\,] & \text{“empty” environment} \\ \mid & \Gamma, x : A & \text{“non-empty” environment} \end{array}$$

typing environments are comma-separated snoc-lists of (variable,type)-pairs – in fact only the lists whose variables are mutually distinct get used

Notation

- Γ ok means that no variable occurs more than once in Γ

Definition ($\lambda^{\to}$ typing relation: $\Gamma \vdash t : A$)

Γ ranges over typing environments (or typing contexts)

$$
\begin{array}{lll}
\Gamma := & & \\
\mid & [\,] & \text{“empty” environment} \\
\mid & \Gamma, x : A & \text{“non-empty” environment}
\end{array}
$$

typing environments are comma-separated snoc-lists of (variable,type)-pairs – in fact only the lists whose variables are mutually distinct get used

Notation

- Γ ok means that no variable occurs more than once in Γ
- dom Γ denotes the finite set of variables occurring in Γ

Definition ($\lambda^{\rightarrow}$ typing relation: $\Gamma \vdash t : A$ (cont'd))

$$\frac{\Gamma\ \text{ok} \qquad x \notin \text{dom}\,\Gamma}{\Gamma, x : A \vdash x : A}\ \text{(var)} \qquad \frac{\Gamma \vdash x : A \qquad x' \notin \text{dom}\,\Gamma}{\Gamma, x' : A \vdash x : A}\ \text{(var')} \qquad \frac{\Gamma\ \text{ok}}{\Gamma \vdash c^A : A}\ \text{(const)}$$

$$\frac{\Gamma\ \text{ok}}{\Gamma \vdash () : \text{unit}}\ \text{(unit)} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash (s, t) : A \times B}\ \text{(pair)} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \text{fst}\ t : A}\ \text{(fstT)}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \text{snd}\ t : B}\ \text{(sndT)} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\, t : A \rightarrow B}\ \text{(fun)} \qquad \frac{\Gamma \vdash s : A \rightarrow B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\ t : B}\ \text{(app)}$$

Example (term examples)

- $[\,] \vdash \lambda z : (A \rightarrow B) \times (A \rightarrow C).\ \lambda x : A.\ ((\text{fst } z)\ x, (\text{snd } z)\ x)$

Example (term examples)

- $[\,] \vdash \lambda z\colon (A \rightarrow B) \times (A \rightarrow C).\, \lambda x\colon A.\, ((\mathrm{fst}\ z)\ x, (\mathrm{snd}\ z)\ x)$ has type $((A \rightarrow B) \times (A \rightarrow C) \rightarrow (A \rightarrow (B \times C)))$

Example (term examples)

- $[\,] \vdash \lambda z{:}\,(A \to B) \times (A \to C).\, \lambda x{:}\,A.\, ((\mathsf{fst}\ z)\ x, (\mathsf{snd}\ z)\ x)$ has type $((A \to B) \times (A \to C) \to (A \to (B \times C)))$
- $[\,] \vdash \lambda z{:}\,A \to (B \times C).\, (\lambda x{:}\,A.\,\mathsf{fst}\ (z\ x), \lambda y{:}\,A.\,\mathsf{snd}\ (z\ y))$ has type

Example (term examples)

- $[\,] \vdash \lambda z\colon (A \rightarrow B) \times (A \rightarrow C).\, \lambda x\colon A.\, ((\mathrm{fst}\ z)\ x, (\mathrm{snd}\ z)\ x)$ has type $((A \rightarrow B) \times (A \rightarrow C) \rightarrow (A \rightarrow (B \times C)))$
- $[\,] \vdash \lambda z\colon A \rightarrow (B \times C).\, (\lambda x\colon A.\, \mathrm{fst}\ (z\ x), \lambda y\colon A.\, \mathrm{snd}\ (z\ y))$ has type $(A \rightarrow (B \times C)) \rightarrow ((A \rightarrow B) \times (A \rightarrow C))$

Example (term examples)

- $[\,] \vdash \lambda z\colon (A \to B) \times (A \to C).\, \lambda x\colon A.\, ((\mathrm{fst}\ z)\ x, (\mathrm{snd}\ z)\ x)$ has type $((A \to B) \times (A \to C) \to (A \to (B \times C)))$
- $[\,] \vdash \lambda z\colon A \to (B \times C).\, (\lambda x\colon A.\, \mathrm{fst}\ (z\ x), \lambda y\colon A.\, \mathrm{snd}\ (z\ y))$ has type $(A \to (B \times C)) \to ((A \to B) \times (A \to C))$
- $[\,] \vdash \lambda z\colon A \to (B \times C).\, \lambda x\colon A.\, ((\mathrm{fst}\ z)\ x, (\mathrm{snd}\ z)\ x)$

Example (term examples)

- $[\,] \vdash \lambda z\colon (A \rightarrow B) \times (A \rightarrow C).\, \lambda x\colon A.\, ((\mathsf{fst}\ z)\ x, (\mathsf{snd}\ z)\ x)$ has type $((A \rightarrow B) \times (A \rightarrow C) \rightarrow (A \rightarrow (B \times C)))$
- $[\,] \vdash \lambda z\colon A \rightarrow (B \times C).\, (\lambda x\colon A.\, \mathsf{fst}\ (z\ x), \lambda y\colon A.\, \mathsf{snd}\ (z\ y))$ has type $(A \rightarrow (B \times C)) \rightarrow ((A \rightarrow B) \times (A \rightarrow C))$
- $[\,] \vdash \lambda z\colon A \rightarrow (B \times C).\, \lambda x\colon A.\, ((\mathsf{fst}\ z)\ x, (\mathsf{snd}\ z)\ x)$ has no type (ill-typed term)

Example (typing derivation)

in a typing context $\Gamma = [\,], f : A \rightarrow B, g : B \rightarrow C$, we have an example derivation of a term $s : A \rightarrow C$ as follows:

$$
\dfrac{
\dfrac{
\dfrac{}{\Gamma \vdash g : B \rightarrow C}\,(\text{var})
}{\Gamma, x : A \vdash g : B \rightarrow C}\,(\text{var'})
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{[\,], f : A \rightarrow B \vdash f : A \rightarrow B}\,(\text{var})
}{\Gamma \vdash f : A \rightarrow B}\,(\text{var'})
}{\Gamma, x : A \vdash f : A \rightarrow B}\,(\text{var'})
\qquad
\dfrac{}{\Gamma, x : A \vdash x : A}\,(\text{var})
}{\Gamma, x : A \vdash f\ x : B}\,(\text{app})
}{
\dfrac{\Gamma, x : A \vdash g\ (f\ x) : C}{\Gamma \vdash \lambda x : A.\, g\ (f\ x) : A \rightarrow C}\,(\text{fun})
}\,(\text{app})
$$

Example (typing derivation)

in a typing context $\Gamma = [\,], f : A \rightarrow B, g : B \rightarrow C$, we have an example derivation of a term $s : A \rightarrow C$ as follows:

$$
\dfrac{
  \dfrac{
    \dfrac{}{\Gamma \vdash g : B \rightarrow C}\;\text{(var)}
  }{\Gamma, x : A \vdash g : B \rightarrow C}\;\text{(var')}
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{}{[\,], f : A \rightarrow B \vdash f : A \rightarrow B}\;\text{(var)}
      }{\Gamma \vdash f : A \rightarrow B}\;\text{(var')}
    }{\Gamma, x : A \vdash f : A \rightarrow B}\;\text{(var')}
    \qquad
    \dfrac{}{\Gamma, x : A \vdash x : A}\;\text{(var)}
  }{\Gamma, x : A \vdash f\ x : B}\;\text{(app)}
}{
  \dfrac{\Gamma, x : A \vdash g\ (f\ x) : C}{\Gamma \vdash \lambda x : A.\, g\ (f\ x) : A \rightarrow C}\;\text{(fun)}
}\;\text{(app)}
$$

Remark

the $\lambda^{\rightarrow}$ typing rules are “syntax-directed”, by the structure of terms t and then in the case of variables x, by the structure of typing environments Γ.

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning
- e.g., $\lambda f : A \rightarrow B.\, \lambda x : A.\, f\ x$ should have the same meaning as $\lambda x : A \rightarrow B.\, \lambda y : A.\, x\ y$

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning
- e.g., $\lambda f : A \to B.\, \lambda x : A.\, f\ x$ should have the same meaning as $\lambda x : A \to B.\, \lambda y : A.\, x\ y$
- this issue is best dealt with at the level of syntax rather than semantics

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning
- e.g., $\lambda f : A \to B.\, \lambda x : A.\, f\ x$ should have the same meaning as $\lambda x : A \to B.\, \lambda y : A.\, x\ y$
- this issue is best dealt with at the level of syntax rather than semantics
- from now on we re-define $\lambda^{\to}$ term to mean not an abstract syntax tree but rather an equivalence class of such trees with respect to α-equivalence $s =_\alpha t$:

$$\frac{}{c^A =_\alpha c^A} \qquad \frac{}{x =_\alpha x} \qquad \frac{}{() =_\alpha ()}$$

$$\frac{s =_\alpha s' \qquad t =_\alpha t'}{(s, t) =_\alpha (s', t')} \qquad \frac{t =_\alpha t'}{\mathrm{fst}\ t =_\alpha \mathrm{fst}\ t'} \qquad \frac{t =_\alpha t'}{\mathrm{snd}\ t =_\alpha \mathrm{snd}\ t'}$$

$$\frac{s =_\alpha s' \qquad t =_\alpha t'}{s\ t =_\alpha s'\ t'} \qquad \frac{t \cdot (y\ x) =_\alpha t' \cdot (y\ x') \qquad y \text{ does not occur in } \{x, x', t, t'\}}{\lambda x : A.\, t =_\alpha \lambda x' : A.\, t'}$$

where $t \cdot (y\ x)$ denotes the result of replacing all occurrences of x with y in t

Example (α-equivalence)

$$\begin{array}{rcccc} \lambda x\!:\!A.\,x\,x & =_\alpha & \lambda y\!:\!A.\,y\,y & \neq_\alpha & \lambda x\!:\!A.\,x\,y \\ (\lambda y\!:\!A.\,y)\;x & =_\alpha & (\lambda x\!:\!A.\,x)\;x & \neq_\alpha & (\lambda x\!:\!A.\,x)\;y \end{array}$$

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x : A.\, \_$ binder) by the term s

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x : A.\_$ binder) by the term s
- alpha-converting λ-bound variables in t to avoid them "capturing" any free variables of t

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x : A.\, \_$ binder) by the term s
- alpha-converting λ-bound variables in t to avoid them "capturing" any free variables of t
- e.g., $(\lambda y : A.\, (y, x))[y/x]$ is $\lambda z : A.\, (z, y)$ and is not $\lambda y : A.\, (y, y)$

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x\colon A.\,\_$ binder) by the term s
- alpha-converting λ-bound variables in t to avoid them "capturing" any free variables of t
- e.g., $(\lambda y\colon A.\,(y,x))[y/x]$ is $\lambda z\colon A.\,(z,y)$ and is not $\lambda y\colon A.\,(y,y)$
- the relation $t[s/x] = t'$ can be inductively defined by the following rules:

$$\frac{}{c^A[s/x] = c^A} \qquad \frac{}{x[s/x] = s} \qquad \frac{y \neq x}{y[s/x] = y} \qquad \frac{}{()[s/x] = ()}$$

$$\frac{t_1[s/x] = t_1' \qquad t_2[s/x] = t_2'}{(t_1, t_2)[s/x] = (t_1', t_2')} \qquad \frac{t[s/x] = t'}{(\mathrm{fst}\ t)[s/x] = \mathrm{fst}\ t'} \qquad \frac{t[s/x] = t'}{(\mathrm{snd}\ t)[s/x] = \mathrm{snd}\ t'} \qquad \frac{t_1[s/x] = t_1' \qquad t_2[s/x] = t_2'}{(t_1\ t_2)[s/x] = t_1'\ t_2'}$$

$$\frac{t[s/x] = t' \qquad y \neq x \text{ and } y \text{ does not freely occur in } s}{(\lambda y\colon A.\,t)[s/x] = \lambda y\colon A.\,t'}$$

Definition ($\beta\eta$-equality)

the relation $\Gamma \vdash s =_{\beta\eta} t : A$ (where Γ ranges over typing environments, s and t over terms and A over types) is inductively defined by the following rules:

- β-conversion

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x : A.\, t)\ s =_{\beta\eta} t[s/x] : B} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{fst}(s, t) =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{snd}(s, t) =_{\beta\eta} t : B}$$

Definition ($\beta\eta$-equality)

the relation $\Gamma \vdash s =_{\beta\eta} t : A$ (where Γ ranges over typing environments, s and t over terms and A over types) is inductively defined by the following rules:

- β-conversion

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x : A.\, t)\ s =_{\beta\eta} t[s/x] : B} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{fst}(s, t) =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{snd}(s, t) =_{\beta\eta} t : B}$$

- η-conversion

$$\frac{\Gamma \vdash t : A \to B \qquad x \text{ does not occur in } t}{\Gamma \vdash t =_{\beta\eta} (\lambda x : A.\, t\ x) : A \to B} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t =_{\beta\eta} (\mathrm{fst}\ t, \mathrm{snd}\ t) : A \times B} \qquad \frac{\Gamma \vdash t : \mathrm{unit}}{\Gamma \vdash t =_{\beta\eta} () : \mathrm{unit}}$$

Definition ($\beta\eta$-equality)

the relation $\Gamma \vdash s =_{\beta\eta} t : A$ (where Γ ranges over typing environments, s and t over terms and A over types) is inductively defined by the following rules:

- β-conversion

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x : A.\, t)\; s =_{\beta\eta} t[s/x] : B} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{fst}(s, t) =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{snd}(s, t) =_{\beta\eta} t : B}$$

- η-conversion

$$\frac{\Gamma \vdash t : A \to B \qquad x \text{ does not occur in } t}{\Gamma \vdash t =_{\beta\eta} (\lambda x : A.\, t\; x) : A \to B} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t =_{\beta\eta} (\mathrm{fst}\; t, \mathrm{snd}\; t) : A \times B} \qquad \frac{\Gamma \vdash t : \mathrm{unit}}{\Gamma \vdash t =_{\beta\eta} () : \mathrm{unit}}$$

- congruence rules

$$\frac{\Gamma, x : A \vdash t =_{\beta\eta} t' : B}{\Gamma \vdash \lambda x : A.\, t =_{\beta\eta} \lambda x : A.\, t' : A \to B} \qquad \frac{\Gamma \vdash s =_{\beta\eta} s' : A \to B \qquad \Gamma \vdash t =_{\beta\eta} t' : A}{\Gamma \vdash s\; t =_{\beta\eta} s'\; t' : B}$$

Definition ($\beta\eta$-equality)

the relation $\Gamma \vdash s =_{\beta\eta} t : A$ (where Γ ranges over typing environments, s and t over terms and A over types) is inductively defined by the following rules:

- β-conversion

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x : A.\, t)\ s =_{\beta\eta} t[s/x] : B} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathsf{fst}(s, t) =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathsf{snd}(s, t) =_{\beta\eta} t : B}$$

- η-conversion

$$\frac{\Gamma \vdash t : A \to B \qquad x \text{ does not occur in } t}{\Gamma \vdash t =_{\beta\eta} (\lambda x : A.\, t\ x) : A \to B} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t =_{\beta\eta} (\mathsf{fst}\ t, \mathsf{snd}\ t) : A \times B} \qquad \frac{\Gamma \vdash t : \mathrm{unit}}{\Gamma \vdash t =_{\beta\eta} () : \mathrm{unit}}$$

- congruence rules

$$\frac{\Gamma, x : A \vdash t =_{\beta\eta} t' : B}{\Gamma \vdash \lambda x : A.\, t =_{\beta\eta} \lambda x : A.\, t' : A \to B} \qquad \frac{\Gamma \vdash s =_{\beta\eta} s' : A \to B \qquad \Gamma \vdash t =_{\beta\eta} t' : A}{\Gamma \vdash s\ t =_{\beta\eta} s'\ t' : B}$$

- $=_{\beta\eta}$ is reflexive, symmetric and transitive

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t =_{\beta\eta} t : A} \qquad \frac{\Gamma \vdash s =_{\beta\eta} t : A}{\Gamma \vdash t =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash r =_{\beta\eta} s : A \qquad \Gamma \vdash s =_{\beta\eta} t : A}{\Gamma \vdash r =_{\beta\eta} t : A}$$

Theorem (Progress)

$\forall e : \texttt{term}, \vdash e : \tau \implies \texttt{value}\ e \lor \exists e', e \to_{\beta} e'$

Theorem (Progress)

$\forall e : \texttt{term}, \vdash e : \tau \implies \texttt{value}\ e \lor \exists e', e \rightarrow_{\beta} e'$

Theorem (Preservation)

$\forall e, e' : \texttt{term}, \vdash e : \tau \land e \rightarrow_{\beta} e' \implies \vdash e' : \tau$

Thanks! $\&$ Questions?