Types in Programming Languages (2)

Christian Urban

`http://www4.in.tum.de/lehre/vorlesungen/types/WS0607/`

Announcements

- No lecture next week (1st November is holiday).
- The lecture on the 8th of November (week after next) will be in “Church”.

Last Week

- extremely simple language:

$$
\begin{aligned}
T &::= \mathsf{bool} \mid \mathsf{nat} \\
e &::= x \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{gr}\ e\ e \mid \mathsf{le}\ e\ e \\
&\quad \mid \mathsf{eq}\ e\ e \mid \mathsf{if}\ e\ e\ e \mid 0 \mid \mathsf{succ}\ e \mid \mathsf{iszero}\ e
\end{aligned}
$$

- Inference rules for typing:

$$
\frac{\mathsf{valid}\ \Gamma \quad (x : T) \in \Gamma}{\Gamma \vdash x : T}
\qquad
\frac{\mathsf{valid}\ \Gamma}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}
$$

$$
\frac{\mathsf{valid}\ \Gamma}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}
$$

Other Typing-Rules

$$\frac{\Gamma \vdash e_1 : \mathsf{nat} \quad \Gamma \vdash e_2 : \mathsf{nat}}{\Gamma \vdash \mathsf{gr}\ e_1\ e_2 : \mathsf{bool}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{nat} \quad \Gamma \vdash e_2 : \mathsf{nat}}{\Gamma \vdash \mathsf{le}\ e_1\ e_2 : \mathsf{bool}}$$

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathsf{eq}\ e_1\ e_2 : \mathsf{bool}}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \mathsf{if}\ e_1\ e_2\ e_3 : T}$$

$$\frac{\mathsf{valid}\ \Gamma}{\Gamma \vdash 0 : \mathsf{nat}} \qquad \frac{\Gamma \vdash e : \mathsf{nat}}{\Gamma \vdash \mathsf{succ}\ e : \mathsf{nat}}$$

$$\frac{\Gamma \vdash e : \mathsf{nat}}{\Gamma \vdash \mathsf{iszero}\ e : \mathsf{bool}}$$

Applying Rules

The typing rules are syntax-directed.

$$\{x : \mathsf{bool}\} \vdash \mathsf{eq}\ x\ (\mathsf{iszero}\ 0) : \mathsf{bool}$$

Applying Rules

The typing rules are syntax-directed.

$$\{x : \mathsf{bool}\} \vdash \mathsf{eq}\ x\ (\mathsf{iszero}\ 0) : \mathsf{bool}$$

$$\boxed{\dfrac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathsf{eq}\ e_1\ e_2 : \mathsf{bool}}}$$

Applying Rules

The typing rules are syntax-directed.

$$\frac{\{x : \mathsf{bool}\} \vdash x : \mathsf{bool} \qquad \{x : \mathsf{bool}\} \vdash \mathsf{iszero}\ 0 : \mathsf{bool}}{\{x : \mathsf{bool}\} \vdash \mathsf{eq}\ x\ (\mathsf{iszero}\ 0) : \mathsf{bool}}$$

$$\boxed{\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathsf{eq}\ e_1\ e_2 : \mathsf{bool}}}$$

Applying Rules

The typing rules are syntax-directed.

$$\dfrac{\dfrac{\text{valid } \{x:\text{bool}\} \qquad (x:\text{bool}) \in \{x:\text{bool}\}}{\{x:\text{bool}\} \vdash x:\text{bool}} \qquad \{x:\text{bool}\} \vdash \text{iszero } 0:\text{bool}}{\{x : \text{bool}\} \vdash \text{eq } x \ (\text{iszero } 0) : \text{bool}}$$

$$\boxed{\dfrac{\text{valid } \mathit{\Gamma} \quad (x : T) \in \mathit{\Gamma}}{\mathit{\Gamma} \vdash x : T}}$$

Applying Rules

The typing rules are syntax-directed.

$$\dfrac{\dfrac{\dfrac{\mathsf{valid}\ \varnothing \quad x \notin \mathsf{dom}\ \varnothing}{\mathsf{valid}\ \{x:\mathsf{bool}\}} \quad (x:\mathsf{bool}) \in \{x:\mathsf{bool}\}}{\{x:\mathsf{bool}\} \vdash x:\mathsf{bool}} \qquad \{x:\mathsf{bool}\} \vdash \mathsf{iszero}\ 0:\mathsf{bool}}{\{x:\mathsf{bool}\} \vdash \mathsf{eq}\ x\ (\mathsf{iszero}\ 0) : \mathsf{bool}}$$

$$\boxed{\dfrac{\mathsf{valid}\ \Gamma \quad x \notin \mathsf{dom}\ \Gamma}{\mathsf{valid}\ (x:T) \cup \Gamma}}$$

Applying Rules

The typing rules are syntax-directed.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{valid}\ \varnothing} \qquad x \notin \mathsf{dom}\ \varnothing}{\mathsf{valid}\ \{x : \mathsf{bool}\}} \qquad (x : \mathsf{bool}) \in \{x : \mathsf{bool}\}}{\{x : \mathsf{bool}\} \vdash x : \mathsf{bool}} \qquad \{x : \mathsf{bool}\} \vdash \mathsf{iszero}\ 0 : \mathsf{bool}$$

$$\overline{\{x : \mathsf{bool}\} \vdash \mathsf{eq}\ x\ (\mathsf{iszero}\ 0) : \mathsf{bool}}$$

$$\boxed{\dfrac{}{\mathsf{valid}\ \varnothing}}$$

Applying Rules

The typing rules are syntax-directed.

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{valid}\ \varnothing} \quad x \notin \mathsf{dom}\ \varnothing}{\mathsf{valid}\ \{x : \mathsf{bool}\}} \quad (x : \mathsf{bool}) \in \{x : \mathsf{bool}\}}{\{x : \mathsf{bool}\} \vdash x : \mathsf{bool}} \qquad \dfrac{\{x : \mathsf{bool}\} \vdash 0 : \mathsf{nat}}{\{x : \mathsf{bool}\} \vdash \mathsf{iszero}\ 0 : \mathsf{bool}}}{\{x : \mathsf{bool}\} \vdash \mathsf{eq}\ x\ (\mathsf{iszero}\ 0) : \mathsf{bool}}
$$

$$
\boxed{\dfrac{\Gamma \vdash e : \mathsf{nat}}{\Gamma \vdash \mathsf{iszero}\ e : \mathsf{bool}}}
$$

Applying Rules

The typing rules are syntax-directed.

$$
\dfrac{
  \dfrac{\dfrac{}{\text{valid } \varnothing} \quad x \notin \text{dom } \varnothing}{\text{valid } \{x\!:\!\text{bool}\}} \quad (x\!:\!\text{bool}) \in \{x\!:\!\text{bool}\}
}{\{x\!:\!\text{bool}\} \vdash x : \text{bool}}
\qquad
\dfrac{
  \dfrac{\text{valid } \{x\!:\!\text{bool}\}}{\{x\!:\!\text{bool}\} \vdash 0 : \text{nat}}
}{\{x\!:\!\text{bool}\} \vdash \text{iszero } 0 : \text{bool}}
$$

$$
\{x : \text{bool}\} \vdash \text{eq } x \ (\text{iszero } 0) : \text{bool}
$$

(The last line is the conclusion under a single rule line spanning both subderivations above.)

$$
\boxed{\dfrac{\text{valid } \Gamma}{\Gamma \vdash 0 : \text{nat}}}
$$

Applying Rules

The typing rules are syntax-directed.

$$
\dfrac{
\dfrac{\dfrac{}{\mathsf{valid}\ \varnothing} \quad x \notin \mathsf{dom}\ \varnothing}{\mathsf{valid}\ \{x:\mathsf{bool}\}} \quad (x:\mathsf{bool}) \in \{x:\mathsf{bool}\}}{\{x:\mathsf{bool}\} \vdash x : \mathsf{bool}}
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{valid}\ \varnothing} \quad x \notin \mathsf{dom}\ \varnothing}{\mathsf{valid}\ \{x:\mathsf{bool}\}}}{\{x:\mathsf{bool}\} \vdash 0 : \mathsf{nat}}}{\{x:\mathsf{bool}\} \vdash \mathsf{iszero}\ 0 : \mathsf{bool}}
}{\{x : \mathsf{bool}\} \vdash \mathsf{eq}\ x\ (\mathsf{iszero}\ 0)\ :\ \mathsf{bool}}
$$

Applying Rules

They are not deterministic.

$$\frac{\{x : \mathsf{bool}\} \vdash x : \mathsf{nat} \qquad \{x : \mathsf{bool}\} \vdash \mathsf{iszero}\ 0 : \mathsf{nat}}{\{x : \mathsf{bool}\} \vdash \mathsf{eq}\ x\ (\mathsf{iszero}\ 0) : \mathsf{bool}}$$

$$\boxed{\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathsf{eq}\ e_1\ e_2 : \mathsf{bool}}}$$

Applying Rules

They are not deterministic.

$$\dfrac{\dfrac{\text{valid } \{x:\text{bool}\} \qquad (x:\text{nat}) \in \{x:\text{bool}\}}{\{x:\text{bool}\} \vdash x:\text{nat}} \qquad \{x:\text{bool}\} \vdash \text{iszero } 0:\text{nat}}{\{x:\text{bool}\} \vdash \text{eq } x \text{ (iszero } 0) : \text{bool}}$$

$$\boxed{\dfrac{\text{valid } \Gamma \quad (x:T) \in \Gamma}{\Gamma \vdash x : T}}$$

Rule Inductions

The general pattern of a rule is:

$$\frac{\text{premise}_1 \ldots \text{premise}_n \quad \text{side-conditions}}{\text{conclusion}}$$

We can show that a property P holds for all elements given by rules, by

- showing that the property holds for the axioms (we can assume the side-conditions)
- holds for the conclusion of all other rules, assuming it holds already for the premises (we can also assume the side-conditions)

Rule Inductions

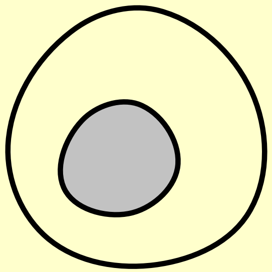
The general pattern of a rule is:

$$\frac{\text{premise}_1 \ldots \text{premise}_n \quad \text{side-conditions}}{\text{conclusion}}$$

We can sl ll
elements

- show e
  axior ons)
- holds for the conclusion of all other rules,
  assuming it holds already for the premises
  (we can also assume the side-conditions)

The rules define a subset:



We showed:

If $\Gamma \vdash e : T$ then valid Γ.

Run-Time

- We said: A strongly typed programming language prevents all forbidden errors.

if (eq 0 true) true false

Run-Time

- We said: *A* strongly typed programming language prevents all forbidden errors.

  if (eq 0 true) true false

- We introduce an evaluation relation:

  $$e \Downarrow v$$

  where e is program; v is value (result of the computation)

Evaluation Relation

- Values v ::= true | false | 0 | succ v
- Rules:

$$\frac{}{\text{true} \Downarrow \text{true}} \qquad \frac{}{\text{false} \Downarrow \text{false}}$$

$$\frac{}{0 \Downarrow 0} \qquad \frac{e \Downarrow v}{\text{succ } e \Downarrow \text{succ } v}$$

$$\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad v_1 > v_2}{\text{gr } e_1\ e_2 \Downarrow \text{true}}$$

$$\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad v_1 \not> v_2}{\text{gr } e_1\ e_2 \Downarrow \text{false}}$$

Evaluation Relation

- Similar for le and eq
- Rules for if and iszero

$$\frac{e_1 \Downarrow \mathsf{true} \quad e_2 \Downarrow v_2}{\mathsf{if}\ e_1\ e_2\ e_3 \Downarrow v_2} \qquad \frac{e_1 \Downarrow \mathsf{false} \quad e_3 \Downarrow v_3}{\mathsf{if}\ e_1\ e_2\ e_3 \Downarrow v_3}$$

$$\frac{e \Downarrow 0}{\mathsf{ifzero}\ e \Downarrow \mathsf{true}} \qquad \frac{e \Downarrow v \quad v \neq 0}{\mathsf{ifzero}\ e \Downarrow \mathsf{false}}$$

Evaluation Relation

- Similar for le and eq
- Rules for if and iszero

$$\frac{e_1 \Downarrow \mathsf{true} \quad e_2 \Downarrow v_2}{\mathsf{if}\ e_1\ e_2\ e_3 \Downarrow v_2} \qquad \frac{e_1 \Downarrow \mathsf{false} \quad e_3 \Downarrow v_3}{\mathsf{if}\ e_1\ e_2\ e_3 \Downarrow v_3}$$

$$\frac{e \Downarrow 0}{\mathsf{ifzero}\ e \Downarrow \mathsf{true}} \qquad \frac{e \Downarrow v \quad v \neq 0}{\mathsf{ifzero}\ e \Downarrow \mathsf{false}}$$

- Note we do not have a rule for variables.

Closed Expressions

- A closed expression does not contain any variables.

Closed Expressions

- A closed expression does not contain any variables.

- Inductive definition:

$$\frac{}{\text{closed true}} \qquad \frac{}{\text{closed false}}$$

$$\frac{\text{closed } e_1 \quad \text{closed } e_2 \quad \text{closed } e_3}{\text{closed (if } e_1\ e_2\ e_3)}$$

Closed Expressions

- A closed expression does not contain any variables.
- Inductive definition:

$$\frac{}{\text{closed true}} \qquad \frac{}{\text{closed false}}$$

$$\frac{\text{closed } e_1 \quad \text{closed } e_2 \quad \text{closed } e_3}{\text{closed } (\text{if } e_1\ e_2\ e_3)}$$

- If e is typable and closed, then there exists a T such that $\varnothing \vdash e : T$.

Closed Expressions

- A closed expression does not contain any variables.

- Inductive definition:

$$\frac{}{\text{closed true}} \qquad \frac{}{\text{closed false}}$$

closed

c

An expression e is typable, provided there exists a Γ and T such that $\Gamma \vdash e : T$.

- If e is typable and closed, then there exists a T such that $\varnothing \vdash e : T$.

Closed Expressions

- A closed expression does not contain any variables.

- Inductive definition:

$$\frac{}{\text{closed true}} \qquad \frac{}{\text{closed false}}$$

$$\frac{\text{closed } e_1 \quad \text{closed } e_2 \quad \text{closed } e_3}{\text{closed } (\text{if } e_1\ e_2\ e_3)}$$

- If e is typable and closed, then there exists a T such that $\varnothing \vdash e : T$.

Structural Induction

$$\forall x.\, P\, x$$
$$P\, \mathsf{true}$$
$$P\, \mathsf{false}$$
$$\forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow$$
$$\forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow$$
$$\forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow$$
$$\forall e_1\, e_2\, e_3.\, P\, e_1 \wedge P\, e_2 \wedge$$
$$P\, 0$$
$$\forall e.\, P\, e \Rightarrow P\, (\mathsf{succ}\, e)$$
$$\forall e.\, P\, e \Rightarrow P\, (\mathsf{iszero}\, e)$$

$$\forall e.\, P\, e$$

$$
\begin{array}{lcl}
e & ::= & x \\
 & \mid & \mathsf{true} \\
 & \mid & \mathsf{false} \\
 & \mid & \mathsf{gr}\; e\; e \\
 & \mid & \mathsf{le}\; e\; e \\
 & \mid & \mathsf{eq}\; e\; e \\
 & \mid & \mathsf{if}\; e\; e\; e \\
 & \mid & 0 \\
 & \mid & \mathsf{succ}\; e \\
 & \mid & \mathsf{iszero}\; e
\end{array}
$$

Structural Induction

$$\frac{\begin{array}{l}\forall x.\, P\, x \\ P\ \mathsf{true} \\ P\ \mathsf{false} \\ \forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow P\, (\mathsf{gr}\ e_1\, e_2) \\ \forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow P\, (\mathsf{le}\ e_1\, e_2) \\ \forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow P\, (\mathsf{eq}\ e_1\, e_2) \\ \forall e_1\, e_2\, e_3.\, P\, e_1 \wedge P\, e_2 \wedge P\, e_3 \Rightarrow P\, (\mathsf{if}\ e_1\, e_2\, e_3) \\ P\, 0 \\ \forall e.\, P\, e \Rightarrow P\, (\mathsf{succ}\ e) \\ \forall e.\, P\, e \Rightarrow P\, (\mathsf{iszero}\ e)\end{array}}{\forall e.\, P\, e}$$

Proof

- Property $P\ e$:

$$(\exists \Gamma T.\ \Gamma \vdash e : T) \wedge \text{closed } e \Rightarrow \exists T.\ \varnothing \vdash e : T$$

- Case $e = x$: We have to show $\forall x. P\ x$.

Proof

- Property $P\ e$:

$$(\exists \Gamma T.\ \Gamma \vdash e : T) \wedge \text{closed } e \Rightarrow \exists T.\ \varnothing \vdash e : T$$

- Case $e = \text{true}$: We have to show P true.

$$\frac{\text{valid } \Gamma}{\Gamma \vdash \text{true} : \text{bool}} \qquad \frac{}{\text{closed true}}$$

Proof

- Property $P\ e$:

$$(\exists \Gamma T.\ \Gamma \vdash e : T) \wedge \mathsf{closed}\ e \Rightarrow \exists T.\ \varnothing \vdash e : T$$

- Case $e = \mathsf{succ}\, e'$: $\forall e'.\, P\ e' \Rightarrow P\ (\mathsf{succ}\, e')$

  We can assume:
  $$(\exists \Gamma T.\ \Gamma \vdash e' : T) \wedge \mathsf{closed}\ e' \Rightarrow \exists T.\ \varnothing \vdash e' : T$$

  We have to show:
  $$(\exists \Gamma T.\ \Gamma \vdash \mathsf{succ}\, e' : T) \wedge \mathsf{closed}\ (\mathsf{succ}\, e') \Rightarrow \exists T.\ \varnothing \vdash \mathsf{succ}\, e' : T$$

Further Properties

- For every closed and typable expression e there exists a value v such that:

$$e \Downarrow v\,.$$

- If $e \Downarrow v_1$ and $e \Downarrow v_2$ then $v_1 = v_2$.

- If $e \Downarrow v$ and $\varnothing \vdash e : T$, then $\varnothing \vdash v : T$.

$$P\,e\,v = \forall T.\, \varnothing \vdash e\!:\!T \Rightarrow \varnothing \vdash v\!:\!T$$

Further Properties

- For every closed and typable expression e there exists a value v such that:

$$e \Downarrow v \, .$$

- If $e \Downarrow v_1$ and $e \Downarrow v_2$ then $v_1 = v_2$.

- If $e \Downarrow v$ and $\varnothing \vdash e : T$, then $\varnothing \vdash v : T$.

$$P\, e\, v = \forall T.\ \varnothing \vdash e : T \Rightarrow \varnothing \vdash v : T$$

Bad News: Language Safety

…consists of two properties

- **Type Preservation**:

  If e is closed, and $\varnothing \vdash e : T$, and $e \Downarrow v$, then $\varnothing \vdash v : T$.

- and **Progress**—well-typed programs cannot get stuck:

  If $\varnothing \vdash e : T$, then either e is a value, or e makes a “transition”.

  Progress is a property which also holds for non-terminating programs.

Transition Relation

- Values $v ::= \mathsf{true} \mid \mathsf{false} \mid 0 \mid \mathsf{succ}\ v$
- Rules:

$$\frac{e \mapsto e'}{\mathsf{succ}\ e \mapsto \mathsf{succ}\ e'}$$

$$\frac{e_1 \mapsto e_1'}{\mathsf{gr}\ e_1\ e_2 \mapsto \mathsf{gr}\ e_1'\ e_2} \qquad \frac{\mathsf{value}\ v \quad e_2 \mapsto e_2'}{\mathsf{gr}\ v\ e_2 \mapsto \mathsf{gr}\ v\ e_2'}$$

$$\frac{\mathsf{value}\ v_1 \quad \mathsf{value}\ v_2 \quad v_1 > v_2}{\mathsf{gr}\ v_1\ v_2 \mapsto \mathsf{true}}$$

Evaluation Relation

- Similar for le and eq
- Rules for if

$$\frac{e_1 \mapsto e_1'}{\mathsf{if}\ e_1\ e_2\ e_3 \mapsto \mathsf{if}\ e_1'\ e_2\ e_3}$$

$$\frac{e_2 \mapsto e_2'}{\mathsf{if}\ \mathsf{true}\ e_2\ e_3 \mapsto \mathsf{if}\ \mathsf{true}\ e_2'\ e_3}$$

$$\frac{\mathsf{value}\ v \quad e_2 \mapsto v}{\mathsf{if}\ \mathsf{true}\ e_2\ e_3 \mapsto v}$$

Evaluation Relation

- Similar for le and eq
- Rules for if

$$\frac{e_1 \mapsto e_1'}{\text{if } e_1\ e_2\ e_3 \mapsto \text{if } e_1'\ e_2\ e_3}$$

$$\frac{e_2 \mapsto e_2'}{\text{if true } e_2\ e_3 \mapsto \text{if true } e_2'\ e_3}$$

$$\frac{\text{value } v \quad e_2 \mapsto v}{\text{if true } e_2\ e_3 \mapsto v}$$

- Note that if $0\ e_1\ e_2$ is stuck.

Properties

- If $e \Downarrow v$ then $e \mapsto \ldots \mapsto v$.
- If $e \mapsto \ldots \mapsto e'$ and e' is a value, then $e \Downarrow e'$.
- If e is closed and well-typed then either e is a value, or there exists an e' such that $e \mapsto e'$.
- If $e \mapsto e'$ and $\varnothing \vdash e : T$, then $\varnothing \vdash e' : T$.

A More Interesting Language

- (Raw) Terms:

$$
\begin{array}{rcll}
e & ::= & x & \text{variables} \\
  & \mid & e\ e & \text{applications} \\
  & \mid & \lambda x.e & \text{lambda-abstractions} \\
  & \mid & \mathsf{let}\ x = e\ \mathsf{in}\ e & \text{lets}
\end{array}
$$

- We introduce next a simple-type system

$$
\begin{array}{rcll}
T & ::= & X & \text{type variables} \\
  & \mid & T \to T & \text{function types}
\end{array}
$$

Simple Type-System

- Variables

$$\frac{\mathsf{valid}\ \Gamma \quad (x : T) \in \Gamma}{\Gamma \vdash x : T}$$

- Applications

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\ e_2 : T_2}$$

- Lambdas

$$\frac{x : T_1, \Gamma \vdash e : T_2 \quad x \notin \mathsf{dom}\, \Gamma}{\Gamma \vdash \lambda x.e : T_1 \to T_2}$$

- Lets

$$\frac{\Gamma \vdash e_1 : T_1 \quad x : T_1, \Gamma \vdash e_2 : T_2 \quad x \notin \mathsf{dom}\, \Gamma}{\Gamma \vdash \mathsf{let}\, x = e_1\, \mathsf{in}\, e_2 : T_2}$$

A Problem

- $\lambda y.\lambda x.x$:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{valid}\ \{x{:}T_2, y{:}T_1\}} \quad (x{:}T_2) \in \{x{:}T_2, y{:}T_1\}}{\{x{:}T_2, y{:}T_1\} \vdash x : T_2} \qquad x \not\in \mathsf{dom}\ \{y{:}T_1\}}{\{y{:}T_1\} \vdash \lambda x.x : T_2 \to T_2} \qquad y \not\in \mathsf{dom}\ \varnothing}{\varnothing \vdash \lambda y.\lambda x.x : T_1 \to T_2 \to T_2}
$$

A Problem

- $\lambda y.\lambda x.x$:

$$
\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \mathsf{valid}\,\{x\!:\!T_2, y\!:\!T_1\}\end{array} \quad (x\!:\!T_2) \in \{x\!:\!T_2, y\!:\!T_1\}}{\{x\!:\!T_2, y\!:\!T_1\} \vdash x : T_2} \qquad x \notin \mathsf{dom}\,\{y\!:\!T_1\}}{\{y\!:\!T_1\} \vdash \lambda x.x : T_2 \to T_2} \qquad y \notin \mathsf{dom}\,\varnothing}{\varnothing \vdash \lambda y.\lambda x.x : T_1 \to T_2 \to T_2}
$$

- $\lambda x.\lambda x.x$:

$$
\dfrac{\dfrac{\{x\!:\!T_2, x\!:\!T_1\} \vdash x : T_2 \qquad \color{red}{x \notin \mathsf{dom}\,\{x\!:\!T_1\}}}{\{x\!:\!T_1\} \vdash \lambda x.x : T_2 \to T_2} \qquad y \notin \mathsf{dom}\,\varnothing}{\varnothing \vdash \lambda x.\lambda x.x : T_1 \to T_2 \to T_2}
$$

A Problem

Given the language

$$
\begin{array}{llll}
e & ::= & x & \text{variables} \\
 & | & e\ e & \text{applications} \\
 & | & \lambda x.e & \text{lambda-abstractions} \\
 & | & \mathsf{let}\ x = e\ \mathsf{in}\ e & \text{lets}
\end{array}
$$

When we define $\Gamma \vdash e : T$ then e stands for an alpha-equivalence class!!!

$$
\dfrac{\dfrac{\{x{:}T_2, x{:}T_1\} \vdash x : T_2 \qquad x \notin \mathsf{dom}\,\{x{:}T_1\}}{\{x{:}T_1\} \vdash \lambda x.x : T_2 \to T_2} \qquad y \notin \mathsf{dom}\,\varnothing}{\varnothing \vdash \lambda x.\lambda x.x : T_1 \to T_2 \to T_2}
$$

Alpha-Equivalence Classes

Bound Variables programs are <u>not</u> just strings of ascii-characters (at least not for people who study programming languages).

```
int sum (int x, int y) {
  return (x+y);
}
```

Alpha-Equivalence Classes

Bound Variables

programs are not just strings of ascii-characters (at least not for people who study programming languages).

```
int sum (int x, int y) {
  return (x+y);
}
```

=

```
int sum (int foo, int bar) {
  return (foo+bar);
}
```

Alpha-Equivalence Classes

Bound Variables programs are <u>not</u> just strings of ascii-characters (at least not for people who study programming languages).

```
int sum (int x, int y) {
  return (x+y);
}
```

=

```
int sum (int foo, int bar) {
  return (foo+bar);
}
```

α**-equivalence** expresses the idea that the names of the bound variables are unimportant

Alpha-Equivalence Classes

Bound Variables programs are not just strings of ascii-characters (at least not for people who study programming languages).

```
int sum (int x, int y) {
  return (x+y);
}
```

$=$

```
int sum (int foo, int bar) {
  return (foo+bar);
}
```

$\int_0^1 x^2 + 1 \, dx \qquad \sum_{x=0}^{10} x^2 + 1 \qquad \exists x.\, x^2 + 1 = 0$

Alpha-Equivalence Classes

Bound Variables programs are <u>not</u> just strings of ascii-characters (at least not for people who study programming languages).

```
int sum (int x, int y) {
  return (x+y);
}
```

=

```
int sum (int foo, int bar) {
  return (foo+bar);
}
```

$$\int_0^1 x^2 + 1\, dx \qquad \sum_{x=0}^{10} x^2 + 1 \qquad \exists x.\, x^2 + 1 = 0$$

Also in the **lambda-calculus**, you have:

$$\lambda x.\lambda y.\, x\ y = \lambda \mathsf{foo}.\lambda \mathsf{bar}.\, \mathsf{foo}\ \mathsf{bar}$$

Free Variables

- (Raw) Terms:

$$
\begin{array}{rcll}
e & ::= & x & \text{variables} \\
 & | & e\ e & \text{applications} \\
 & | & \lambda x.e & \text{lambda-abstractions} \\
 & | & \mathsf{let}\ x = e\ \mathsf{in}\ e & \text{lets}
\end{array}
$$

- free variables:

$$
\begin{array}{rcl}
\mathsf{fv}(x) & \stackrel{\mathsf{def}}{=} & \{x\} \\
\mathsf{fv}(e_1\, e_2) & \stackrel{\mathsf{def}}{=} & \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \\
\mathsf{fv}(\lambda x.e) & \stackrel{\mathsf{def}}{=} & \mathsf{fv}(e) - \{x\} \\
\mathsf{fv}(\mathsf{let}\, x = e_1\, \mathsf{in}\, e_2) & \stackrel{\mathsf{def}}{=} & (\mathsf{fv}(e_2) - \{x\}) \cup \mathsf{fv}(e_1)
\end{array}
$$

Swapping

- $$e ::= x \quad \text{variables}$$
  $$\mid\ e\ e \quad \text{applications}$$
  $$\mid\ \lambda x.e \quad \text{lambda-abstractions}$$
  $$\mid\ \mathsf{let}\ x = e\ \mathsf{in}\ e \quad \text{lets}$$

e	$::=$	x	variables
	$\mid$	$e\ e$	applications
	$\mid$	$\lambda x.e$	lambda-abstractions
	$\mid$	$\mathsf{let}\ x = e\ \mathsf{in}\ e$	lets

- A swapping operation:

$$(x\,y)\bullet z \stackrel{\mathsf{def}}{=} \begin{cases} y & \text{if } z = x \\ x & \text{if } z = y \\ z & \text{o'wise} \end{cases}$$

$$(x\,y)\bullet(e_1\,e_2) \stackrel{\mathsf{def}}{=} ((x\,y)\bullet e_1)\ ((x\,y)\bullet e_2)$$

$$(x\,y)\bullet(\lambda z.e) \stackrel{\mathsf{def}}{=} \lambda((x\,y)\bullet z).((x\,y)\bullet e)$$

$$(x\,y)\bullet(\mathsf{let}\ z = e_1\ \mathsf{in}\ e_2) \stackrel{\mathsf{def}}{=} \mathsf{let}\ (x\,y)\bullet z = (x\,y)\bullet e_1\ \mathsf{in}\ (x\,y)\bullet e_2$$

Swapping

- $e \quad ::= \quad x$ variables
  $\quad | \quad e\ e$ applications
  $\quad | \quad \lambda x.e$ lambda-abstractions
  $\quad | \quad \mathsf{let}\ x = e\ \mathsf{in}\ e$ lets

- A swapp We have $(x\,y)\bullet(x\,y)\bullet e = e$.

$$(x\,y)\bullet z \stackrel{\mathrm{def}}{=} \begin{cases} y & \text{if } z = x \\ x & \text{if } z = y \\ z & \text{o'wise} \end{cases}$$

$$(x\,y)\bullet(e_1\,e_2) \stackrel{\mathrm{def}}{=} ((x\,y)\bullet e_1)\ ((x\,y)\bullet e_2)$$

$$(x\,y)\bullet(\lambda z.e) \stackrel{\mathrm{def}}{=} \lambda((x\,y)\bullet z).((x\,y)\bullet e)$$

$$(x\,y)\bullet(\mathsf{let}\ z = e_1\ \mathsf{in}\ e_2) \stackrel{\mathrm{def}}{=} \mathsf{let}\ (x\,y)\bullet z = (x\,y)\bullet e_1\ \mathsf{in}\ (x\,y)\bullet e_2$$

Alpha-Equivalence

$$\frac{}{x \approx x} \qquad \frac{e_1 \approx s_1 \quad e_2 \approx s_2}{e_1\, e_2 \approx s_1\, s_2}$$

$$\frac{e \approx s}{\lambda x.e \approx \lambda x.s} \qquad \frac{x \neq y \quad e \approx (x\, y)\bullet s \quad x \notin \mathsf{fv}(s)}{\lambda x.e \approx \lambda y.s}$$

$$\frac{e_1 \approx s_1 \quad e_2 \approx s_2}{\mathsf{let}\, x = e_1\, \mathsf{in}\, e_2 \approx \mathsf{let}\, x = s_1\, \mathsf{in}\, s_2}$$

$$\frac{x \neq y \quad e_1 \approx s_1 \quad e_2 \approx (x\, y)\bullet s_2 \quad x \notin \mathsf{fv}(s_2)}{\mathsf{let}\, x = e_1\, \mathsf{in}\, e_2 \approx \mathsf{let}\, y = s_1\, \mathsf{in}\, s_2}$$

Alpha-Equivalence

$$e_1 \approx s_1 \quad e_2 \approx s_2$$

- (reflexivity) $e \approx e$
- (symmetry) if $e_1 \approx e_2$ then $e_2 \approx e_1$
- (transitivity) if $e_1 \approx e_2$ and $e_2 \approx e_3$ then $e_1 \approx e_3$

λx

$$\frac{e_1 \approx s_1 \quad e_2 \approx s_2}{\mathsf{let}\, x = e_1\, \mathsf{in}\, e_2 \approx \mathsf{let}\, x = s_1\, \mathsf{in}\, s_2}$$

$$\frac{x \neq y \quad e_1 \approx s_1 \quad e_2 \approx (x\,y)\bullet s_2 \quad x \not\in \mathsf{fv}(s_2)}{\mathsf{let}\, x = e_1\, \mathsf{in}\, e_2 \approx \mathsf{let}\, y = s_1\, \mathsf{in}\, s_2}$$

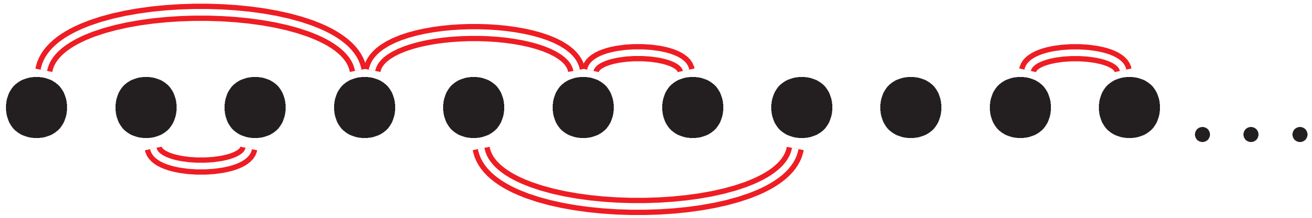
Alpha-Equivalence Classes

Assume we have all (raw) lambda-terms:

$$\bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet \ldots$$

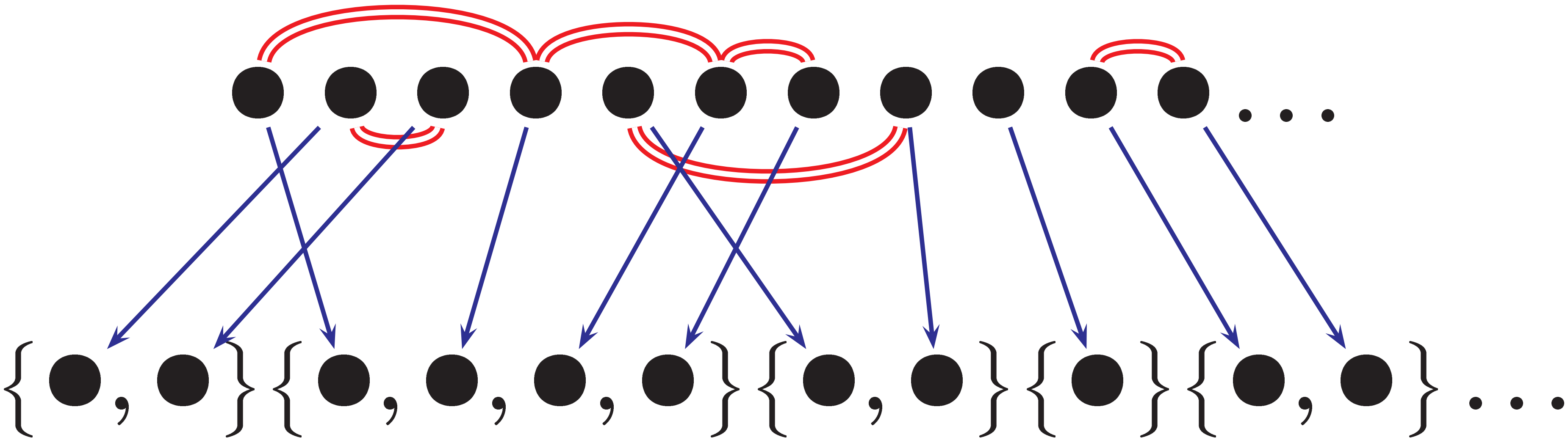
Alpha-Equivalence Classes

Assume we have all (raw) lambda-terms:



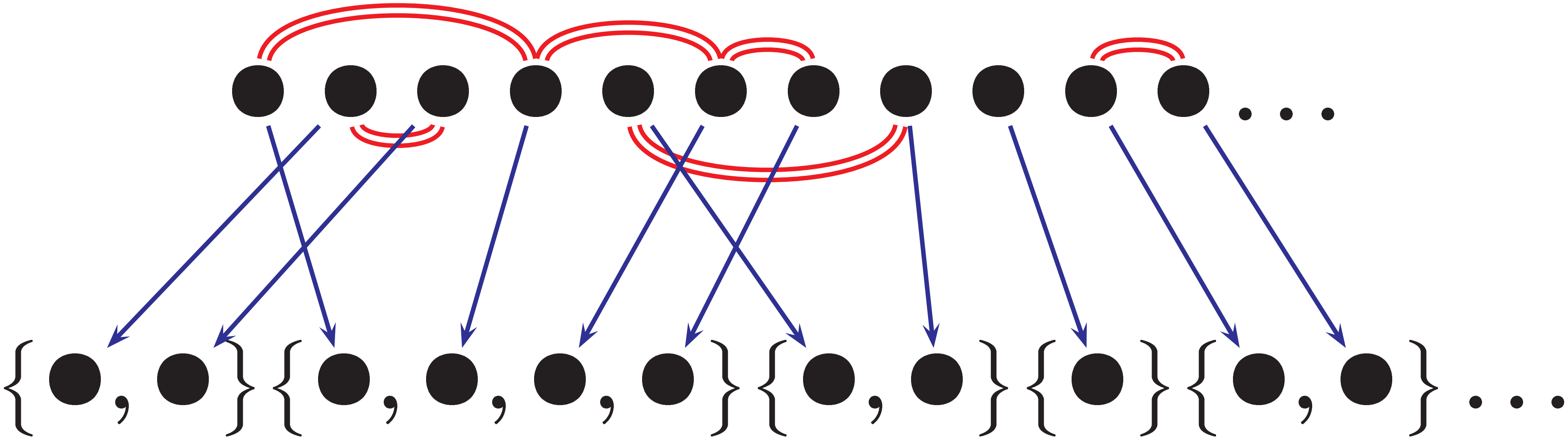
Alpha-Equivalence Classes

Assume we have all (raw) lambda-terms:



Alpha-Equivalence Classes

Assume we have all (raw) lambda-terms:



$$
\begin{aligned}
[x]_\alpha &= \{x' \mid x \approx x'\} = \{x\} \\
[e_1\, e_2]_\alpha &= \{e_1'\, e_2' \mid e_1\, e_2 \approx e_1'\, e_2'\} \\
[\lambda x.e]_\alpha &= \{\lambda x'.e' \mid \lambda x.e \approx \lambda x'\, e'\}
\end{aligned}
$$

Simple Type-System

- Variables

$$\frac{\mathsf{valid}\ \Gamma \quad (x : T) \in \Gamma}{\Gamma \vdash [x]_\alpha : T}$$

- Applications

$$\frac{\Gamma \vdash [e_1]_\alpha : T_1 \to T_2 \quad \Gamma \vdash [e_2]_\alpha : T_1}{\Gamma \vdash [e_1\ e_2]_\alpha : T_2}$$

- Lambdas

$$\frac{x : T_1, \Gamma \vdash [e]_\alpha : T_2 \quad x \notin \mathsf{dom}\,\Gamma}{\Gamma \vdash [\lambda x.e]_\alpha : T_1 \to T_2}$$

- Lets

$$\frac{\Gamma \vdash [e_1]_\alpha : T_1 \quad x : T_1, \Gamma \vdash [e_2]_\alpha : T_2 \quad x \notin \mathsf{dom}\,\Gamma}{\Gamma \vdash [\mathsf{let}\, x = e_1\ \mathsf{in}\, e_2]_\alpha : T_2}$$

Simple Type-System

- Variables

$$\frac{\mathsf{valid}\ \Gamma \quad (x : T) \in \Gamma}{\Gamma \vdash x : T}$$

- Applications

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\ e_2 : T_2}$$

- Lambdas

$$\frac{x : T_1, \Gamma \vdash e : T_2 \quad x \notin \mathsf{dom}\ \Gamma}{\Gamma \vdash \lambda x.e : T_1 \to T_2}$$

- Lets

$$\frac{\Gamma \vdash e_1 : T_1 \quad x : T_1, \Gamma \vdash e_2 : T_2 \quad x \notin \mathsf{dom}\ \Gamma}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : T_2}$$

The Problem Goes Away

- $\lambda x.\lambda x.x$:

$$\dfrac{\dfrac{}{\{x\!:\!T_1\} \vdash [\lambda x.x]_\alpha : T_2 \rightarrow T_2} \qquad x \notin \mathsf{dom}\,\varnothing}{\varnothing \vdash [\lambda x.\lambda x.x]_\alpha : T_1 \rightarrow T_2 \rightarrow T_2}$$

The Problem Goes Away

- $\lambda x.\lambda x.x$:

$$\dfrac{\dfrac{}{\{x\!:\!T_1\} \vdash [\lambda y.y]_\alpha : T_2 \to T_2} \qquad x \notin \mathrm{dom}\,\varnothing}{\varnothing \vdash [\lambda x.\lambda x.x]_\alpha : T_1 \to T_2 \to T_2}$$

$$[\lambda x.x]_\alpha = [\lambda y.y]_\alpha$$

The Problem Goes Away

- $\lambda x.\lambda x.x$:

$$
\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \mathsf{valid}\,\{y\!:\!T_2, x\!:\!T_1\} \\ (y\!:\!T_2) \in \{y\!:\!T_2, x\!:\!T_1\}\end{array}}{\{y\!:\!T_2, x\!:\!T_1\} \vdash [y]_\alpha : T_2} \qquad y \notin \mathsf{dom}\,\{x\!:\!T_1\}}{\{x\!:\!T_1\} \vdash {\color{red}[\lambda y.y]_\alpha} : T_2 \to T_2} \qquad x \notin \mathsf{dom}\,\varnothing}{\varnothing \vdash [\lambda x.\lambda x.x]_\alpha : T_1 \to T_2 \to T_2}
$$

A Property

- Weakening Lemma

  If $\Gamma_1 \vdash e : T$ and $\Gamma_1 \subseteq \Gamma_2$ and $\text{valid}\ \Gamma_2$ then $\Gamma_2 \vdash e : T$.

- By induction over $\Gamma_1 \vdash e : T$.

$$P\,\Gamma_1\,e\,T = \forall \Gamma_2.\ \Gamma_1 \subseteq \Gamma_2 \wedge \text{valid}\ \Gamma_2 \Rightarrow \Gamma_2 \vdash e : T$$

In the literature the proof of this lemma is often labelled as "trivial", "obvious" etc.

Variable Case

- Rule

$$\frac{\text{valid}\ \Gamma \quad (x:T) \in \Gamma}{\Gamma \vdash x : T}$$

- We have to show for the conclusion:

$$P\ \Gamma\ x\ T = \forall \Gamma_2.\ \Gamma \subseteq \Gamma_2 \wedge \text{valid}\ \Gamma_2 \Rightarrow \Gamma_2 \vdash x : T$$

we know	we have to show
$(x:T) \in \Gamma$ $\Gamma \subseteq \Gamma_2$ valid Γ_2	$\Gamma_2 \vdash x : T$

Lambda Case

- $$\frac{x : T_1, \Gamma \vdash e : T_2 \quad x \notin \mathrm{dom}\, \Gamma}{\Gamma \vdash \lambda x.e : T_1 \to T_2}$$
- We have to show for the conclusion:

$$P\,\Gamma\,(\lambda x.e)\;T_1 \to T_2 =$$
$$\forall \Gamma_2.\ \Gamma \subseteq \Gamma_2 \wedge \mathrm{valid}\,\Gamma_2 \Rightarrow \Gamma_2 \vdash (\lambda x.e) : T_1 \to T_2$$

- We can assume for the premise:

$$P\,(x{:}T_1, \Gamma)\;e\;T_2 =$$
$$\forall \Gamma_2.\ (x{:}T_1, \Gamma) \subseteq \Gamma_2 \wedge \mathrm{valid}\,\Gamma_2 \Rightarrow \Gamma_2 \vdash e : T_2$$

- and we know $x \notin \mathrm{dom}\, \Gamma$

A Proof that Works

- Extend swapping to contexts

$$(x\,y)\bullet\Gamma \stackrel{\text{def}}{=} \{((x\,y)\bullet z : T) \mid (z : T) \in \Gamma\}$$

- We have valid $(x\,y)\bullet\Gamma$ iff valid Γ!!
- We can show for every x and y

$$\Gamma \vdash e : T \Rightarrow ((x\,y)\bullet\Gamma) \vdash (x\,y)\bullet e : T$$

A Proof that Works

- Extend swapping to contexts

$$(x\,y)\bullet\Gamma \stackrel{\text{def}}{=} \{((x\,y)\bullet z : T) \mid (z : T) \in \Gamma\}$$

- We have valid $(x\,y)\bullet\Gamma$ iff valid Γ!!
- We can show for every x and y

$$\Gamma \vdash e : T \Rightarrow ((x\,y)\bullet\Gamma) \vdash (x\,y)\bullet e : T$$

$$((x\,y)\bullet\Gamma) \vdash (x\,y)\bullet e : T \Rightarrow \Gamma \vdash e : T$$

A Proof that Works

- Extend swapping to contexts

$$(x\,y)\bullet\Gamma \stackrel{\mathsf{def}}{=} \{((x\,y)\bullet z : T) \mid (z : T) \in \Gamma\}$$

- We have valid $(x\,y)\bullet\Gamma$ iff valid Γ!!
- We can show for every x and y

$$\Gamma \vdash e : T \Rightarrow ((x\,y)\bullet\Gamma) \vdash (x\,y)\bullet e : T$$

$$((x\,y)\bullet\Gamma) \vdash (x\,y)\bullet e : T \Rightarrow \Gamma \vdash e : T$$

$$P\,\Gamma_1\,e\,T \Rightarrow P\,(x\,y)\bullet\Gamma_1\,(x\,y)\bullet e\,T$$

Lambda Case Again

- We have $x \notin \mathrm{dom}\,\Gamma$.
- We have to show for the conclusion:

  $$P\,\Gamma\,(\lambda x.e)\;T_1 \to T_2 = \\ \quad \forall \Gamma_2.\;\Gamma \subseteq \Gamma_2 \wedge \mathrm{valid}\,\Gamma_2 \Rightarrow \Gamma_2 \vdash (\lambda x.e) : T_1 \to T_2$$

  which means we know $\Gamma \subseteq \Gamma_2$, $\mathrm{valid}\,\Gamma_2$ and have to show $\Gamma_2 \vdash (\lambda x.e) : T_1 \to T_2$.
- We choose a fresh y (fresh for $x\,\Gamma\,\Gamma_2\,e$).
- This means $\lambda x.e = \lambda y.(x\,y)\bullet e$ and $(x\,y)\bullet\Gamma = \Gamma$.
- We show $\Gamma_2 \vdash (\lambda y.(x\,y)\bullet e) : T_1 \to T_2$.
- It suffices to show $y\!:\!T_1, \Gamma_2 \vdash (x\,y)\bullet e : T_2$ and $y \notin \mathrm{dom}\,\Gamma_2$.

Lambda Case Again

- What remains is to show $y\!:\!T_1, \Gamma_2 \vdash (x\,y)\bullet e : T_2$.
- We can assume $P\;(x\!:\!T_1,\Gamma)\;e\;T_2$.
- Which means we also have $P\;(y\!:\!T_1,(x\,y)\bullet\Gamma)\;(x\,y)\bullet e\;T_2$ which is equal to $P\;(y\!:\!T_1,\Gamma)\;(x\,y)\bullet e\;T_2$.

$$
\begin{array}{l}
P\;(y\!:\!T_1,\Gamma)\;(x\,y)\bullet e\;T_2 = \\
\quad \forall \Gamma_2.\; (y\!:\!T_1,\Gamma) \subseteq \Gamma_2 \wedge \text{valid}\,\Gamma_2 \Rightarrow \Gamma_2 \vdash (x\,y)\bullet e : T_2
\end{array}
$$

- We instantiate Γ_2 with $y\!:\!T_1,\Gamma_2$. So we have to show $(y\!:\!T_1,\Gamma) \subseteq (y\!:\!T_1,\Gamma_2)$ and $\text{valid}\,(y\!:\!T_1,\Gamma_2)$.

  We can assume $\Gamma \subseteq \Gamma_2$ and $\text{valid}\,\Gamma_2$.
- Done.

Possible Questions

- Give the axioms and rules for inductively generating typing judgements $\Gamma \vdash e : T$, where e ranges over expressions built up from variables using only lambda abstraction $\lambda x.e$, function application $e_1\ e_2$ and local declarations $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$. As part of your answer you should explain what it means for an expression to be alpha-equivalent.

More Next Week

- Slides at the end of

  http://www4.in.tum.de/lehre/vorlesungen/types/WS0607/

  There is also an appraisal form where you can complain anonymously.

- You can say whether the lecture was too easy, too quiet, too hard to follow, too chaotic and so on. You can also comment on things I should repeat.