*x*ML^F, an explicit language for ML^F

Who? Boris Yakobowski

Where? CNRS - University Paris 7

When? February 8, 2009

Outline

Outline

ML^{F}

ML-like type inference +
expressivity of System F second-order polymorphism

$\mathsf{ML^F}$

ML-like type inference +
expressivity of System F second-order polymorphism

Two difficulties:

- Type inference for System F is undecidable
- System F does not have principal types

ML^F

ML-like type inference +
expressivity of System F second-order polymorphism

Two difficulties:

- Type inference for System F is undecidable
- System F does not have principal types

Example:

$$\begin{array}{rcll} \mathsf{id} & \triangleq & \lambda(x)\ x & : \ \forall\beta.\ \beta \to \beta \\ \mathsf{choose} & \triangleq & \lambda(x)\ \lambda(y)\ x & : \ \forall\alpha.\ \alpha \to \alpha \to \alpha \end{array}$$

$$\mathsf{choose\ id} : \begin{cases} (\forall\beta.\ \beta \to \beta) \to (\forall\beta.\ \beta \to \beta) & \alpha = \forall\beta.\ \beta \to \beta \\ \forall\gamma.\ (\gamma \to \gamma) \to (\gamma \to \gamma) & \alpha = \gamma \to \gamma \end{cases}$$

No type is more general than the other

ML^F types: going beyond System F

- To solve the problem of non-principality:

Flexible quantification

ML^F types extend System F types with an instance-bounded quantification of the form $\forall\,(\alpha \geqslant \tau)\;\tau'$:

- Both τ and τ' can be instantiated inside $\forall\,(\alpha \geqslant \tau)\;\tau'$
- All occurrences of α in τ' must pick the same instance of τ

$$
\begin{array}{rcl}
\text{choose id} & : & \forall\,(\alpha \geqslant \forall\beta.\ \beta \to \beta)\ \alpha \to \alpha \\
 & \sqsubseteq & (\forall\beta.\ \beta \to \beta) \to (\forall\beta.\ \beta \to \beta) \\
\text{or} & \sqsubseteq & \forall\gamma.\ (\gamma \to \gamma) \to (\gamma \to \gamma)
\end{array}
$$

ML^F types: going beyond System F

- To solve the problem of non-principality:

Flexible quantification

ML^F types extend System F types with an instance-bounded quantification of the form $\forall\,(\alpha \geqslant \tau)\;\tau'$:

- Both τ and τ' can be instantiated inside $\forall\,(\alpha \geqslant \tau)\;\tau'$
- All occurrences of α in τ' must pick the same instance of τ

- To permit type inference:

Rigid quantification

Instance-bounded quantification, of the form $\forall\,(\alpha = \tau)\;\tau'$

- τ cannot (really) be instantiated inside $\forall\,(\alpha = \tau)\;\tau'$
- But $\forall\,(\alpha = \tau)\;\alpha \to \alpha$ and $\forall\,(\alpha = \tau)\,\forall\,(\alpha' = \tau)\;\alpha \to \alpha'$ are different as far as type inference is concerned

ML^F as a type system

Extends ML and System F, and combines the benefits of both

Compared to ML

- The expressivity of second-order polymorphism is available
- All ML programs remain typable unchanged

Compared to System F

- ML^F has type inference
- Programs (given their type annotations) have principal types

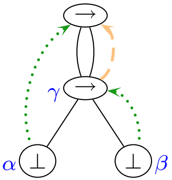
Moreover:

- in practice, programs require very few type annotations
- typable programs are stable under a wide range of program transformations

Graphic MLF types

The superposition of:

- A term-dag, representing the skeleton of the type
- A binding tree, indicating where variables are bound
  Two kind of binding edges, for flexible and rigid quantification

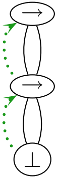


$$\forall(\alpha \geqslant \bot)\,\forall(\gamma = \forall(\beta \geqslant \bot)\,\alpha \to \beta)\,\gamma \to \gamma$$

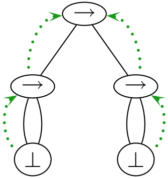
Graphic ML^F types

The superposition of:

- A term-dag, representing the skeleton of the type
- A binding tree, indicating where variables are bound
  Two kind of binding edges, for flexible and rigid quantification
- Sharing of nodes is important



$\neq$



$\forall\,(\alpha \geqslant \sigma_{id})\ \alpha \to \alpha$

$\forall\,(\alpha \geqslant \sigma_{id})\ \forall\,(\beta \geqslant \sigma_{id})\ \alpha \to \beta$

Possible type for $\lambda(x)\ x$

Incorrect for $\lambda(x)\ x$

Instance on graphic ML^F types

The instance relation $\sqsubseteq$

- Four atomic operations on graphs:

Instance on graphic $\mathrm{ML^F}$ types

The instance relation $\sqsubseteq$

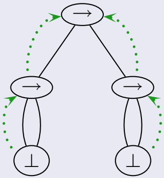
- Four atomic operations on graphs:
  - Grafting: replacing a variable by a closed type (variable substitution)

$\sqsubseteq$

Instance on graphic ML^F types

The instance relation $\sqsubseteq$

- Four atomic operations on graphs:
  - Grafting: replacing a variable by a closed type (variable substitution)
  - Merging: fusing two identical subgraphs (correlates the two corresponding subtypes)



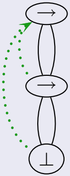
$\sqsubseteq$

Instance on graphic ML^F types

The instance relation $\sqsubseteq$

- Four atomic operations on graphs:
  - Grafting: replacing a variable by a closed type (variable substitution)
  - Merging: fusing two identical subgraphs (correlates the two corresponding subtypes)
  - Raising: edge extrusion (removes the possibility to introduce universal quantification)

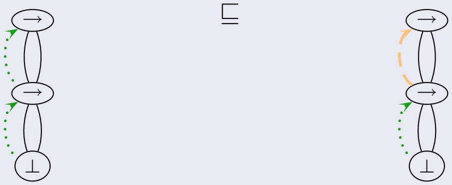
$\sqsubseteq$



Instance on graphic ML^F types

The instance relation $\sqsubseteq$

- Four atomic operations on graphs:
  - Grafting: replacing a variable by a closed type (variable substitution)
  - Merging: fusing two identical subgraphs (correlates the two corresponding subtypes)
  - Raising: edge extrusion (removes the possibility to introduce universal quantification)
  - Weakening: turns a flexible edge into a rigid one (forbids further instantiation of the corresponding type)



Graphic constraints

- Used to formalize the ML^F typing relation, and type inference
- Graphic types extended with four new constructs
  - Unification edges
    Force two nodes to be equal
  - Existential nodes
    "Floating" nodes, used only to introduce other constraints
  - Generalization nodes G
  - Instantiation edges

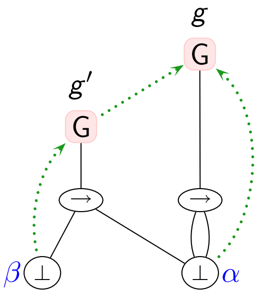
Type generalization

- Type generalization is essential in ML^F, just as in ML
- Gen nodes are used to promote types into type schemes, and to delimit generalization scopes

$g : \ \forall\alpha.\ \alpha \to \alpha$

Type generalization

- Type generalization is essential in ML^F, just as in ML
- Gen nodes are used to promote types into type schemes, and to delimit generalization scopes



$g: \quad \forall\alpha.\ \alpha \to \alpha$

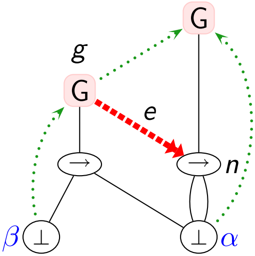
$g': \quad \forall\beta.\ \beta \to \alpha$

α is free at the level of g'

Instantiation edges

- Constrain a node to be an instance of a type scheme

Example:

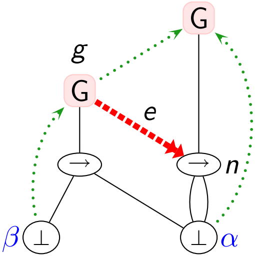


- e constrains n to be an instance of g

Instantiation edges

- Constrain a node to be an instance of a type scheme

Example:



$g: \forall\beta.\ \beta \to \alpha$

$n: \alpha \to \alpha$

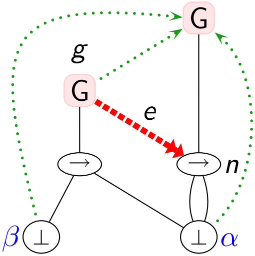
e is solved (take $\beta = \alpha$)

- e constrains n to be an instance of g

Instantiation edges

- Constrain a node to be an instance of a type scheme

Example:

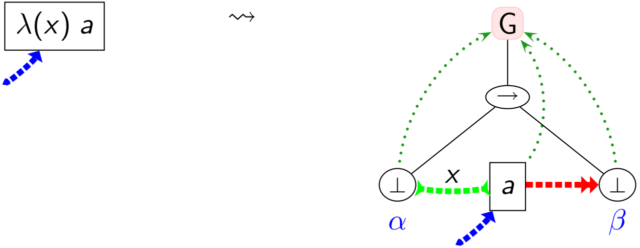


$g: \quad \beta \to \alpha$
$n: \quad \alpha \to \alpha$

e is not solved $\quad (\beta \neq \alpha)$

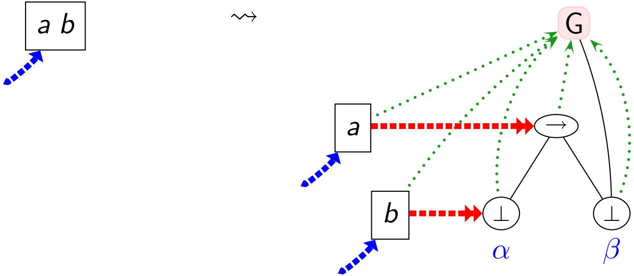
- *e* constrains *n* to be an instance of *g*

Typing constraint for an abstraction



- $\lambda(x)$ *a* can receive type $\alpha \to \beta$, provided
  - α is the (common) type of all the occurrences of *x* in *a*
  - β is an instance of the type of *a*.

Typing constraint for an application



- $a\ b$ can receive type β, provided there exists α such that
  - $a \to \beta$ is an instance of the type of a
  - α is an instance of the type of b

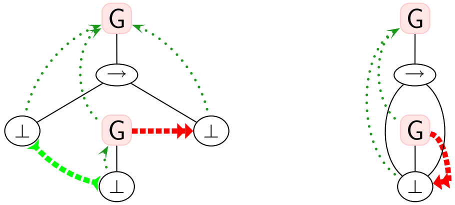
Semantics of constraints

Presolutions

A presolution of a constraint χ is an instance of χ in which all the instantiation and unification edges are solved.

Presolutions retain the shape of the original constraint

Example: Constraint for $\lambda(x)\ x$



Outline

An explicit langage for ML^F

- Study subject reduction in ML^F
  - Type annotations are important inside terms
  - But how to reduce $(e : \sigma)$?

- How to use ML^F inside a typed compiler?
  - ML^F types are more expressive than F ones
  - System F cannot be used as a target langage
    (prior work by Leijen, but not completely satisfactory)

Hence the need for a core, Church-style, langage for ML^F, xML^F

From System F to $x\mathsf{ML^F}$

$x\mathsf{ML^F}$ generalizes System F

- Types: $\sigma \ ::= \ \bot \ | \ \forall\,(\alpha \geqslant \sigma)\,\sigma \ | \ \alpha \ | \ \sigma \to \sigma$

Rigid quantification is only needed for type inference, and is inlined in $x\mathsf{ML^F}$
Hence $\forall\,(\alpha = \sigma)\,\alpha \to \alpha$ becomes $\sigma \to \sigma$

From System F to $x\mathsf{ML}^\mathsf{F}$

$x\mathsf{ML}^\mathsf{F}$ generalizes System F

- Types: $\sigma \;::=\; \bot \;\mid\; \forall\,(\alpha \geqslant \sigma)\,\sigma \;\mid\; \alpha \;\mid\; \sigma \to \sigma$

Rigid quantification is only needed for type inference, and is inlined in $x\mathsf{ML}^\mathsf{F}$
Hence $\forall\,(\alpha = \sigma)\;\alpha \to \alpha$ becomes $\sigma \to \sigma$

- Terms : $a \;::=\; x \;\mid\; \lambda(x : \sigma)\,a \;\mid\; a\;a \;\mid\; \mathsf{let}\;x = a\;\mathsf{in}\;a$
  $\qquad\qquad \mid\; \Lambda(\alpha \geqslant \sigma)\,a \;\mid\; a[\varphi]$

- Typing rules are the same as in System F, except for type application

$$\textsc{TApp}\quad \frac{\Gamma \vdash a : \sigma \qquad \Gamma \vdash \varphi : \sigma \leq \sigma'}{\Gamma \vdash a[\varphi] : \sigma'}$$

Type computations

Instance is explicitely witnessed through the use of type computations

$$\varphi \ ::= \ \varepsilon \ \mid \ \varphi ; \varphi \ \mid \ \triangleright \sigma \ \mid \ \alpha \triangleleft \ \mid \ \forall(\geqslant \varphi) \ \mid \ \forall(\alpha \geqslant)\, \varphi \ \mid \ \& \ \mid \ \otimes$$

Inst-Reflex
$$\frac{}{\Gamma \vdash \varepsilon : \sigma \leq \sigma}$$

Inst-Trans
$$\frac{\Gamma \vdash \varphi_1 : \sigma_1 \leq \sigma_2 \qquad \Gamma \vdash \varphi_2 : \sigma_2 \leq \sigma_3}{\Gamma \vdash \varphi_1 ; \varphi_2 : \sigma_1 \leq \sigma_3}$$

Inst-Bot
$$\frac{}{\Gamma \vdash \triangleright \sigma : \bot \leq \sigma}$$

Inst-Hyp
$$\frac{\alpha \geqslant \sigma \in \Gamma}{\Gamma \vdash \alpha \triangleleft : \sigma \leq \alpha}$$

Inst-Inner
$$\frac{\Gamma \vdash \varphi : \sigma_1 \leq \sigma_2}{\Gamma \vdash \forall(\geqslant \varphi) : \forall(\alpha \geqslant \sigma_1)\, \sigma \leq \forall(\alpha \geqslant \sigma_2)\, \sigma}$$

Inst-Outer
$$\frac{\Gamma, \varphi : \alpha \geqslant \sigma \vdash \varphi : \sigma_1 \leq \sigma_2}{\Gamma \vdash \forall(\alpha \geqslant)\, \varphi : \forall(\alpha \geqslant \sigma)\, \sigma_1 \leq \forall(\alpha \geqslant \sigma)\, \sigma_2}$$

Inst-Quant-Elim
$$\frac{}{\Gamma \vdash \& : \forall(\alpha \geqslant \sigma)\, \sigma' \leq \sigma'\{\alpha \leftarrow \sigma\}}$$

Inst-Quant-Intro
$$\frac{\alpha \notin \mathsf{ftv}(\sigma)}{\Gamma \vdash \otimes : \sigma \leq \forall(\alpha \geqslant \bot)\, \sigma}$$

Type computations

Instance is explicitely witnessed through the use of type computations

$$\varphi \ ::= \ \varepsilon \mid \varphi;\varphi \mid \triangleright\sigma \mid \alpha\triangleleft \mid \forall(\geqslant\varphi) \mid \forall(\alpha\geqslant)\,\varphi \mid \& \mid \otimes$$

Inst-Reflex
$$\frac{}{\Gamma \vdash \varepsilon : \sigma \le \sigma}$$

Inst-Trans
$$\frac{\Gamma \vdash \varphi_1 : \sigma_1 \le \sigma_2 \qquad \Gamma \vdash \varphi_2 : \sigma_2 \le \sigma_3}{\Gamma \vdash \varphi_1;\varphi_2 : \sigma_1 \le \sigma_3}$$

Inst-Bot
$$\frac{}{\Gamma \vdash \triangleright\sigma : \bot \le \sigma}$$

Inst-Hyp
$$\frac{\alpha \geqslant \sigma \in \Gamma}{\Gamma \vdash \alpha\triangleleft : \sigma \le \alpha}$$

Inst-Inner
$$\frac{\Gamma \vdash \varphi : \sigma_1 \le \sigma_2}{\Gamma \vdash \forall(\geqslant\varphi) : \forall(\alpha\geqslant\sigma_1)\,\sigma \le \forall(\alpha\geqslant\sigma_2)\,\sigma}$$

Inst-Outer
$$\frac{\Gamma, \varphi : \alpha \geqslant \sigma \vdash \varphi : \sigma_1 \le \sigma_2}{\Gamma \vdash \forall(\alpha\geqslant)\,\varphi : \forall(\alpha\geqslant\sigma)\,\sigma_1 \le \forall(\alpha\geqslant\sigma)\,\sigma_2}$$

Inst-Quant-Elim
$$\frac{}{\Gamma \vdash \& : \forall(\alpha\geqslant\sigma)\,\sigma' \le \sigma'\{\alpha \leftarrow \sigma\}}$$

Inst-Quant-Intro
$$\frac{\alpha \notin \mathsf{ftv}(\sigma)}{\Gamma \vdash \otimes : \sigma \le \forall(\alpha\geqslant\bot)\,\sigma}$$

Type computations

Instance is explicitely witnessed through the use of type computations

$$\varphi \ ::= \ \varepsilon \mid \varphi;\varphi \mid \triangleright\sigma \mid \alpha\triangleleft \mid \forall(\geqslant\varphi) \mid \forall(\alpha\geqslant)\,\varphi \mid \& \mid \otimes$$

$$\textsc{Inst-Reflex}\quad \frac{}{\Gamma \vdash \varepsilon : \sigma \leq \sigma}$$

$$\textsc{Inst-Trans}\quad \frac{\Gamma \vdash \varphi_1 : \sigma_1 \leq \sigma_2 \qquad \Gamma \vdash \varphi_2 : \sigma_2 \leq \sigma_3}{\Gamma \vdash \varphi_1;\varphi_2 : \sigma_1 \leq \sigma_3}$$

$$\textsc{Inst-Bot}\quad \frac{}{\Gamma \vdash \triangleright\sigma : \bot \leq \sigma}$$

$$\textsc{Inst-Hyp}\quad \frac{\alpha \geqslant \sigma \in \Gamma}{\Gamma \vdash \alpha\triangleleft : \sigma \leq \alpha}$$

$$\textsc{Inst-Inner}\quad \frac{\Gamma \vdash \varphi : \sigma_1 \leq \sigma_2}{\Gamma \vdash \forall(\geqslant\varphi) : \forall(\alpha \geqslant \sigma_1)\,\sigma \leq \forall(\alpha \geqslant \sigma_2)\,\sigma}$$

$$\textsc{Inst-Outer}\quad \frac{\Gamma, \varphi : \alpha \geqslant \sigma \vdash \varphi : \sigma_1 \leq \sigma_2}{\Gamma \vdash \forall(\alpha\geqslant)\,\varphi : \forall(\alpha \geqslant \sigma)\,\sigma_1 \leq \forall(\alpha \geqslant \sigma)\,\sigma_2}$$

$$\textsc{Inst-Quant-Elim}\quad \frac{}{\Gamma \vdash \& : \forall(\alpha \geqslant \sigma)\,\sigma' \leq \sigma'\{\alpha \leftarrow \sigma\}}$$

$$\textsc{Inst-Quant-Intro}\quad \frac{\alpha \notin \mathsf{ftv}(\sigma)}{\Gamma \vdash \otimes : \sigma \leq \forall(\alpha \geqslant \bot)\,\sigma}$$

Type computations

Instance is explicitely witnessed through the use of type computations

$$\varphi ::= \varepsilon \mid \varphi;\varphi \mid \triangleright\sigma \mid \alpha\triangleleft \mid \forall(\geqslant\varphi) \mid \forall(\alpha\geqslant)\,\varphi \mid \& \mid ⅋$$

Inst-Reflex
$$\frac{}{\Gamma \vdash \varepsilon : \sigma \leq \sigma}$$

Inst-Trans
$$\frac{\Gamma \vdash \varphi_1 : \sigma_1 \leq \sigma_2 \qquad \Gamma \vdash \varphi_2 : \sigma_2 \leq \sigma_3}{\Gamma \vdash \varphi_1;\varphi_2 : \sigma_1 \leq \sigma_3}$$

Inst-Bot
$$\frac{}{\Gamma \vdash \triangleright\sigma : \bot \leq \sigma}$$

Inst-Hyp
$$\frac{\alpha \geqslant \sigma \in \Gamma}{\Gamma \vdash \alpha\triangleleft : \sigma \leq \alpha}$$

Inst-Inner
$$\frac{\Gamma \vdash \varphi : \sigma_1 \leq \sigma_2}{\Gamma \vdash \forall(\geqslant\varphi) : \forall(\alpha \geqslant \sigma_1)\,\sigma \leq \forall(\alpha \geqslant \sigma_2)\,\sigma}$$

Inst-Outer
$$\frac{\Gamma, \varphi : \alpha \geqslant \sigma \vdash \varphi : \sigma_1 \leq \sigma_2}{\Gamma \vdash \forall(\alpha\geqslant)\,\varphi : \forall(\alpha \geqslant \sigma)\,\sigma_1 \leq \forall(\alpha \geqslant \sigma)\,\sigma_2}$$

Inst-Quant-Elim
$$\frac{}{\Gamma \vdash \& : \forall(\alpha \geqslant \sigma)\,\sigma' \leq \sigma'\{\alpha \leftarrow \sigma\}}$$

Inst-Quant-Intro
$$\frac{\alpha \notin \mathsf{ftv}(\sigma)}{\Gamma \vdash ⅋ : \sigma \leq \forall(\alpha \geqslant \bot)\,\sigma}$$

Type computations

Instance is explicitely witnessed through the use of type computations

$$\varphi \ ::= \ \varepsilon \mid \varphi;\varphi \mid \triangleright\sigma \mid \alpha\triangleleft \mid \forall(\geqslant\varphi) \mid \forall(\alpha\geqslant)\,\varphi \mid \& \mid \aleph$$

$$\textsc{Inst-Reflex}\quad \frac{}{\Gamma\vdash\varepsilon:\sigma\leq\sigma}$$

$$\textsc{Inst-Trans}\quad \frac{\Gamma\vdash\varphi_1:\sigma_1\leq\sigma_2 \qquad \Gamma\vdash\varphi_2:\sigma_2\leq\sigma_3}{\Gamma\vdash\varphi_1;\varphi_2:\sigma_1\leq\sigma_3}$$

$$\textsc{Inst-Bot}\quad \frac{}{\Gamma\vdash\triangleright\sigma:\bot\leq\sigma}$$

$$\textsc{Inst-Hyp}\quad \frac{\alpha\geqslant\sigma\in\Gamma}{\Gamma\vdash\alpha\triangleleft:\sigma\leq\alpha}$$

$$\textsc{Inst-Inner}\quad \frac{\Gamma\vdash\varphi:\sigma_1\leq\sigma_2}{\Gamma\vdash\forall(\geqslant\varphi):\forall(\alpha\geqslant\sigma_1)\,\sigma\leq\forall(\alpha\geqslant\sigma_2)\,\sigma}$$

$$\textsc{Inst-Outer}\quad \frac{\Gamma,\varphi:\alpha\geqslant\sigma\vdash\varphi:\sigma_1\leq\sigma_2}{\Gamma\vdash\forall(\alpha\geqslant)\,\varphi:\forall(\alpha\geqslant\sigma)\,\sigma_1\leq\forall(\alpha\geqslant\sigma)\,\sigma_2}$$

$$\textsc{Inst-Quant-Elim}\quad \frac{}{\Gamma\vdash\&:\forall(\alpha\geqslant\sigma)\,\sigma'\leq\sigma'\{\alpha\leftarrow\sigma\}}$$

$$\textsc{Inst-Quant-Intro}\quad \frac{\alpha\notin\mathsf{ftv}(\sigma)}{\Gamma\vdash\aleph:\sigma\leq\forall(\alpha\geqslant\bot)\,\sigma}$$

Example: back to choose id

$$\begin{array}{rcll} \text{choose} & \triangleq & \Lambda(\alpha \geqslant \bot)\ \lambda(x:\alpha)\ \lambda(y:\alpha)\ x & :\ \forall\,(\alpha \geqslant \bot)\ \alpha \to \alpha \to \alpha \\ \text{id} & \triangleq & \Lambda(\beta \geqslant \bot)\ \lambda(x:\beta)\ x & :\ \forall\,(\beta \geqslant \bot)\ \beta \to \beta \end{array}$$

- To make choose id well-typed, we must choose a type into which α must be instantiated

Example: back to choose id

$$\begin{aligned} \text{choose} &\triangleq \Lambda(\alpha \geqslant \bot)\, \lambda(x:\alpha)\, \lambda(y:\alpha)\, x : \forall(\alpha \geqslant \bot)\, \alpha \to \alpha \to \alpha \\ \text{id} &\triangleq \Lambda(\beta \geqslant \bot)\, \lambda(x:\beta)\, x : \forall(\beta \geqslant \bot)\, \beta \to \beta \end{aligned}$$

- To make choose id well-typed, we must choose a type into which α must be instantiated

- $$e \triangleq \Lambda(\gamma \geqslant \sigma_{id})\ \underbrace{(\text{choose}[\forall(\geqslant \triangleright \gamma); \&])}_{\gamma\to\gamma\to\gamma}\ \underbrace{(\text{id}[\gamma \triangleleft])}_{\gamma} : \forall(\gamma \geqslant \sigma_{id})\, \gamma \to \gamma$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \triangleright\gamma : \bot \leq \gamma}\ \textsc{Bot}}{\vdash \forall(\geqslant \triangleright \gamma) : \forall(\alpha \geqslant \bot)\, \alpha \to \alpha \to \alpha \leq \forall(\alpha \geqslant \gamma)\, \alpha \to \alpha \to \alpha}\ \textsc{Inner} \qquad \dfrac{}{\vdash \& : \forall(\alpha \geqslant \gamma)\, \alpha \to \alpha \to \alpha \leq \gamma \to \gamma \to \gamma}\ \textsc{Quant-Elim}}{\vdash \forall(\geqslant \triangleright \gamma); \& : \forall(\alpha \geqslant \bot)\, \alpha \to \alpha \to \alpha \leq \gamma \to \gamma \to \gamma}\ \textsc{Trans}$$

Example: back to choose id

$$\begin{aligned} \text{choose} &\triangleq \Lambda(\alpha \geqslant \bot)\, \lambda(x:\alpha)\, \lambda(y:\alpha)\, x : \forall(\alpha \geqslant \bot)\, \alpha \to \alpha \to \alpha \\ \text{id} &\triangleq \Lambda(\beta \geqslant \bot)\, \lambda(x:\beta)\, x \qquad\qquad : \forall(\beta \geqslant \bot)\, \beta \to \beta \end{aligned}$$

- To make choose id well-typed, we must choose a type into which α must be instantiated

- $$e \triangleq \Lambda(\gamma \geqslant \sigma_{id})\, \underbrace{(\text{choose}[\forall(\geqslant \triangleright \gamma); \&])}_{\gamma\to\gamma\to\gamma}\, \underbrace{(\text{id}[\gamma \triangleleft])}_{\gamma} : \forall(\gamma \geqslant \sigma_{id})\, \gamma \to \gamma$$

- We can recover the other System F types just by instantiation

$$\begin{cases} e[\&] & : \sigma_{id} \to \sigma_{id} \\ e[\otimes; \forall(\delta \geqslant)\, (\forall(\geqslant \forall(\geqslant \triangleright \delta); \&); \&)] & : \forall(\delta \geqslant \bot)\, (\delta \to \delta) \to (\delta \to \delta) \end{cases}$$

Reducing expressions

- Usual β-reduction

$$
\begin{array}{lcll}
(\lambda(x:\tau)\ a_1)\ a_2 & \longrightarrow & a_1\{x \leftarrow a_2\} & (\beta) \\
\text{let } x = a_2 \text{ in } a_1 & \longrightarrow & a_1\{x \leftarrow a_2\} & (\beta_{\text{LET}})
\end{array}
$$

Reducing expressions

- Usual β-reduction
- 6 specific rules to reduce type applications

$$
\begin{array}{rcll}
(\lambda(x:\tau)\ a_1)\ a_2 & \longrightarrow & a_1\{x \leftarrow a_2\} & (\beta) \\
\text{let } x = a_2 \text{ in } a_1 & \longrightarrow & a_1\{x \leftarrow a_2\} & (\beta_{\text{LET}}) \\
a[\varepsilon] & \longrightarrow & a & \text{REFLEX} \\
a[\varphi; \varphi'] & \longrightarrow & a[\varphi][\varphi'] & \text{TRANS} \\
a[\otimes] & \longrightarrow & \Lambda(\alpha \geqslant \bot)\ a & \text{QUANT-INTRO} \\
 & & \text{if } \alpha \notin \text{ftv}(a) & \\
(\Lambda(\alpha \geqslant \tau)\ a)[\forall(\alpha \geqslant)\ \varphi] & \longrightarrow & \Lambda(\alpha \geqslant \tau)\ (a[\varphi]) & \text{OUTER} \\
(\Lambda(\alpha \geqslant \tau)\ a)[\forall(\geqslant \varphi)] & \longrightarrow & \Lambda(\alpha \geqslant \tau[\varphi])\ a\{\alpha\triangleleft \leftarrow \varphi; \alpha\triangleleft\} & \text{INNER} \\
(\Lambda(\alpha \geqslant \tau)\ a)[\&] & \longrightarrow & a\{\alpha\triangleleft \leftarrow \varepsilon\}\{\alpha \leftarrow \tau\} & \text{QUANT-ELIM}
\end{array}
$$

Reducing expressions

- Usual β-reduction
- 6 specific rules to reduce type applications
- Context rule $E ::= \{\cdot\} \mid E[\varphi] \mid \lambda(x:\tau)\ E \mid \Lambda(\alpha \geqslant \tau)\ E \mid E\ a \mid a\ E \mid \text{let } x = E \text{ in } a \mid \text{let } x = a \text{ in } E$

$$
\begin{array}{rcll}
(\lambda(x:\tau)\ a_1)\ a_2 & \longrightarrow & a_1\{x \leftarrow a_2\} & (\beta) \\
\text{let } x = a_2 \text{ in } a_1 & \longrightarrow & a_1\{x \leftarrow a_2\} & (\beta_{\text{LET}}) \\
a[\varepsilon] & \longrightarrow & a & \text{REFLEX} \\
a[\varphi;\varphi'] & \longrightarrow & a[\varphi][\varphi'] & \text{TRANS} \\
a[\otimes] & \longrightarrow & \Lambda(\alpha \geqslant \bot)\ a & \text{QUANT-INTRO} \\
 & & \text{if } \alpha \notin \text{ftv}(a) & \\
(\Lambda(\alpha \geqslant \tau)\ a)[\forall(\alpha \geqslant)\ \varphi] & \longrightarrow & \Lambda(\alpha \geqslant \tau)\ (a[\varphi]) & \text{OUTER} \\
(\Lambda(\alpha \geqslant \tau)\ a)[\forall(\geqslant \varphi)] & \longrightarrow & \Lambda(\alpha \geqslant \tau[\varphi])\ a\{\alpha\triangleleft \leftarrow \varphi; \alpha\triangleleft\} & \text{INNER} \\
(\Lambda(\alpha \geqslant \tau)\ a)[\&] & \longrightarrow & a\{\alpha\triangleleft \leftarrow \varepsilon\}\{\alpha \leftarrow \tau\} & \text{QUANT-ELIM} \\
E\{a\} & \longrightarrow & E\{a'\} & \text{CONTEXT} \\
 & & \text{if } a \longrightarrow a' &
\end{array}
$$

Rules Inner and Quant-Elim

- $$(\Lambda(\alpha \geqslant \tau)\ a)[\forall\,(\geqslant \varphi)] \quad\longrightarrow\quad \Lambda(\alpha \geqslant \tau[\varphi])\ a\ ?$$
  $$(\Lambda(\alpha \geqslant \tau)\ a)[\&] \quad\longrightarrow\quad a\{\alpha \leftarrow \tau\}\ ?$$

This is incorrect: after the reduction, the computations $\alpha \triangleleft$ inside a make incorrect assumptions on the bound of α

Rules Inner and Quant-Elim

- $$\begin{aligned}(\Lambda(\alpha \geqslant \tau)\; a)[\forall\,(\geqslant \varphi)] &\longrightarrow \Lambda(\alpha \geqslant \tau[\varphi])\; a\{\alpha \triangleleft \leftarrow \varphi; \alpha \triangleleft\} \\ (\Lambda(\alpha \geqslant \tau)\; a)[\&] &\longrightarrow a\{\alpha \triangleleft \leftarrow \varepsilon\}\{\alpha \leftarrow \tau\}\end{aligned}$$

  This is incorrect: after the reduction, the computations $\alpha \triangleleft$ inside a make incorrect assumptions on the bound of α

- We change those computations:
  - For Inner, $\alpha \triangleleft$ assumed that the bound of α was τ, while it is $\tau[\varphi]$
  - For Quant-Elim, α is now τ, the computations $\alpha \triangleleft$ are vacuous

Example of reductions

- choose id:

$$
\begin{array}{ll}
 & \Lambda(\gamma \geqslant \sigma_{id})\ ((\Lambda(\alpha \geqslant \bot)\ \lambda(x:\alpha)\ \lambda(y:\alpha)\ x)[\forall\,(\geqslant \triangleright \gamma);\, \&])\ (\mathsf{id}[\gamma \triangleleft]) \\
\longrightarrow & \Lambda(\gamma \geqslant \sigma_{id})\ ((\Lambda(\alpha \geqslant \gamma)\ \lambda(x:\alpha)\ \lambda(y:\alpha)\ x)[\&])\ (\mathsf{id}[\gamma \triangleleft]) \\
\longrightarrow & \Lambda(\gamma \geqslant \sigma_{id})\ (\lambda(x:\gamma)\ \lambda(y:\gamma)\ x))\ (\mathsf{id}[\gamma \triangleleft]) \\
\longrightarrow & \Lambda(\gamma \geqslant \sigma_{id})\ \lambda(y:\gamma)\ (\mathsf{id}[\gamma \triangleleft])
\end{array}
$$

- (choose id)[&]:

$$
\begin{array}{ll}
 & (\Lambda(\gamma \geqslant \sigma_{id})\ \lambda(y:\gamma)\ (\mathsf{id}[\gamma \triangleleft]))[\&] \\
\longrightarrow & \lambda(x:\sigma_{id})\ (\mathsf{id}[\epsilon]) \\
\longrightarrow & \lambda(x:\sigma_{id})\ \mathsf{id}
\end{array}
$$

Example of reductions

- choose id:

$$
\begin{array}{ll}
& \Lambda(\gamma \geqslant \sigma_{id}) \; ((\Lambda(\alpha \geqslant \bot) \; \lambda(x:\alpha) \; \lambda(y:\alpha) \; x)[\forall\,(\geqslant \triangleright \gamma); \&]) \; (\mathrm{id}[\gamma \triangleleft]) \\
\longrightarrow & \Lambda(\gamma \geqslant \sigma_{id}) \; ((\Lambda(\alpha \geqslant \gamma) \; \lambda(x:\alpha) \; \lambda(y:\alpha) \; x)[\&]) \; (\mathrm{id}[\gamma \triangleleft]) \\
\longrightarrow & \Lambda(\gamma \geqslant \sigma_{id}) \; (\lambda(x:\gamma) \; \lambda(y:\gamma) \; x)) \; (\mathrm{id}[\gamma \triangleleft]) \\
\longrightarrow & \Lambda(\gamma \geqslant \sigma_{id}) \; \lambda(y:\gamma) \; (\mathrm{id}[\gamma \triangleleft])
\end{array}
$$

- (choose id)[&]:

$$
\begin{array}{ll}
& (\Lambda(\gamma \geqslant \sigma_{id}) \; \lambda(y:\gamma) \; (\mathrm{id}[\gamma \triangleleft]))[\&] \\
\longrightarrow & \lambda(x:\sigma_{id}) \; (\mathrm{id}[\epsilon]) \\
\longrightarrow & \lambda(x:\sigma_{id}) \; \mathrm{id}
\end{array}
$$

- System F like type application $\quad [\tau] \;\triangleq\; [\forall\,(\geqslant \triangleright \tau); \&]$

$$
\begin{aligned}
(\Lambda(\alpha) \; a)[\tau] = (\Lambda(\alpha \geqslant \bot) \; a)[\forall\,(\geqslant \triangleright \tau); \&] \\
\longrightarrow (\Lambda(\alpha \geqslant \tau) \; a)[\&] \\
\longrightarrow a\{\alpha \leftarrow \tau\}
\end{aligned}
$$

$\Rightarrow$ Exactly as in System F

Confluence of strong reduction

- Strong reduction is confluent

proven by the usual method of parallel reductions

Confluence of strong reduction

- Strong reduction is confluent

  proven by the usual method of parallel reductions

- But only on well-typed terms:

$$e \quad \triangleq \quad \big(\Lambda(\alpha \geqslant \forall\,(\gamma)\,\gamma)\ \big((\Lambda(\beta \geqslant \mathsf{int})\ x)[\forall\,(\geqslant \alpha \triangleleft)]\big)\big)[\forall\,(\geqslant \&)]$$

Ill-typed because the computation $\alpha \triangleleft$ is applied to int, while α is supposed to be $\forall\,(\gamma)\,\gamma$

$$\begin{array}{lll} e & \longrightarrow & \big(\Lambda(\alpha \geqslant \forall\,(\gamma)\,\gamma)\ \Lambda(\beta \geqslant \alpha)\ x\big)[\forall\,(\geqslant \&)] \\ & \longrightarrow & \Lambda(\alpha \geqslant \bot)\ \Lambda(\beta \geqslant \alpha)\ x \end{array}$$

(Reducing the innermost type application first, then the outermost)

$$e \quad \longrightarrow \quad \Lambda(\alpha \geqslant \bot)\ \big((\Lambda(\beta \geqslant \mathsf{int})\ x)[\forall\,(\geqslant \&;\alpha \triangleleft)]\big)$$

(Reducing the outermost type application first)

Correctness

- Subject reduction, under any context (including under λ and Λ)
- Progress for call-by-value, with or without the value restriction, and for call-by-name

  First time that ML^F is proven sound for call-by-name

Correctness

- Subject reduction, under any context (including under λ and Λ)
- Progress for call-by-value, with or without the value restriction, and for call-by-name

  First time that ML^{F} is proven sound for call-by-name
- Mechanized proof?
  - almost completed on a previous version of the system, in which ε, $\triangleright\tau$ and $\alpha\triangleleft$ were merged; but need for renaming lemmas
  - $\varphi ::= \alpha\triangleleft \mid \ldots$ not very practical with the locally nameless approach
  - Operation $\varphi\{\alpha\triangleleft \leftarrow \ldots\}$ non standard
  - Boring !

Alias bounds

- In the syntactic presentations of $\mathsf{ML^F}$, $\lambda(x)\ x$ can receive the type

$$\tau \quad \triangleq \quad \forall\,(\alpha \geqslant \bot)\,\forall\,(\beta \geqslant \alpha)\ \beta \to \alpha$$

which is equivalent to $\forall\,(\alpha \geqslant \bot)\ \alpha \to \alpha$

- In $x\mathsf{ML^F}$, $\tau \leq \tau'' \to \tau'$, for any τ' and τ'' such that $\vdash \varphi : \tau' \leq \tau''$ (as witnessed by $\forall\,(\geqslant \triangleright \tau); \&; \forall\,(\geqslant \varphi); \&$)

Alias bounds

- In the syntactic presentations of ML^F, $\lambda(x)\ x$ can receive the type

$$\tau \quad \triangleq \quad \forall\,(\alpha \geqslant \bot)\,\forall\,(\beta \geqslant \alpha)\ \beta \to \alpha$$

which is equivalent to $\forall\,(\alpha \geqslant \bot)\ \alpha \to \alpha$

- In xML^F, $\tau \leq \tau'' \to \tau'$, for any τ' and τ'' such that $\vdash \varphi : \tau' \leq \tau''$ (as witnessed by $\forall\,(\geqslant \triangleright \tau); \&; \forall\,(\geqslant \varphi); \&$)

  Those types are in general incorrect for the identity!

- Thankfully, $\lambda(x)\ x$ cannot receive type τ in xML^F.

- Still, xML^F types are (strictly) more expressive than the usual syntactic ML^F types

Outline

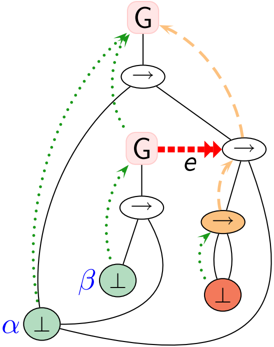
From presolutions to $x\mathsf{ML}^{\mathsf{F}}$ terms

- ML^{F} presolutions can be algorithmically translated into $x\mathsf{ML}^{\mathsf{F}}$ terms

From presolutions to xML^F terms

- ML^F presolutions can be algorithmically translated into xML^F terms
  - Nodes flexibly bound on gen nodes are translated into xML^F type abstractions
  - The fact that an instantiation edge is solved is translated into a type computation
- A bit of care is needed during the translation:
  - presolutions must be slightly normalized
  - order between quantifiers is important in xML^F
  - some differences between the instance relations of ML^F and xML^F

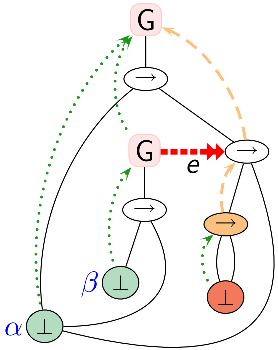
From presolutions to $x\text{ML}^\text{F}$ terms: example



A presolution for $K \triangleq \lambda(x)\ \lambda(y)\ x$

Here, $K : \forall\,(\alpha)\ \alpha \to \sigma_{id} \to \alpha$

From presolutions to xML^F terms: example

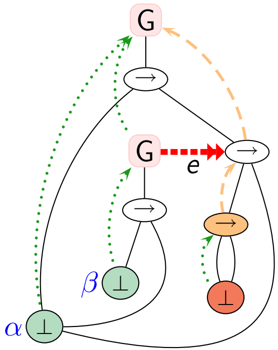


A presolution for $K \triangleq \lambda(x)\ \lambda(y)\ x$

Here, $K : \forall\,(\alpha)\ \alpha \to \sigma_{id} \to \alpha$

$$\Lambda(\alpha)\ \lambda(x:\alpha)\ \underbrace{\underbrace{(\Lambda(\beta)\ \lambda(y:\beta)\ x)}_{\forall\,(\beta)\ \beta\to\alpha}\qquad\qquad\qquad}_{\sigma_{id}\to\alpha}$$

From presolutions to $x\mathsf{ML^F}$ terms: example



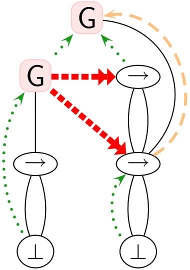
A presolution for $K \triangleq \lambda(x)\ \lambda(y)\ x$

Here, $K : \forall\,(\alpha)\ \alpha \to \sigma_{id} \to \alpha$

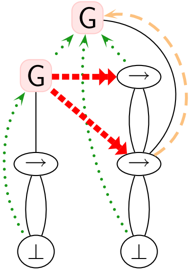
$$\Lambda(\alpha)\ \lambda(x:\alpha)\ \underbrace{\underbrace{(\Lambda(\beta)\ \lambda(y:\beta)\ x)}_{\forall\,(\beta)\,\beta\to\alpha}\ \overbrace{[\forall\,(\geqslant \triangleright \sigma_{id});\,\&]}^{\mathcal{T}(e)}}_{\sigma_{id}\to\alpha}$$

Gen nodes and xMLF terms

- **Example:** id id



$\mathsf{id}[\forall\,(\alpha)\ \alpha \to \alpha]\ \mathsf{id}$



$\Lambda(\alpha)\ (\mathsf{id}[\alpha \to \alpha])\ (\mathsf{id}[\alpha])$

- Nodes bound on the successor of a gen node represent second-order polymorphism kept local
- Nodes bound on a gen node are monomorphic, but re-generalized

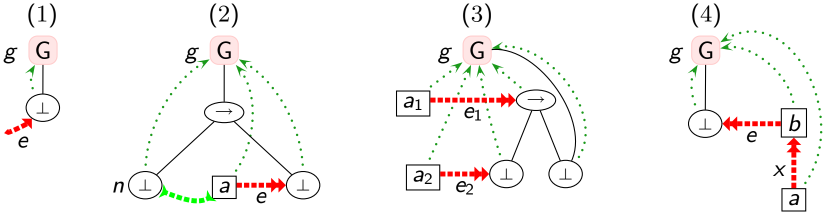
Elaborating λ-terms

$$[\![x]\!] = \begin{cases} x & \text{if } x \text{ is } \lambda\text{-bound} \\ \bigwedge\!\!\!\!\bigwedge(g)\ (x[\mathcal{T}(e)]) & \text{if } x \text{ is let-bound} \end{cases} \quad (1)$$

$$[\![\lambda(x)\ a]\!] = \bigwedge\!\!\!\!\bigwedge(g)\ \lambda(x : \mathrm{Typ}(n))\ ([\![a]\!][\mathcal{T}(e)]) \quad (2)$$

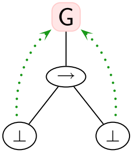
$$[\![a_1\ a_2]\!] = \bigwedge\!\!\!\!\bigwedge(g)\ ([\![a_1]\!][\mathcal{T}(e_1)])\ ([\![a_2]\!][\mathcal{T}(e_2)]) \quad (3)$$

$$[\![\text{let } x = a \text{ in } b]\!] = \bigwedge\!\!\!\!\bigwedge(g)\ \text{let } x = [\![a]\!] \text{ in } ([\![b]\!][\mathcal{T}(e)]) \quad (4)$$



Computing $\bigwedge(g)$

- We add a type quantification for all the nodes flexibly bound on g
  - But in which order?



$$\forall (\alpha) \forall (\beta)\ \alpha \rightarrow \beta$$

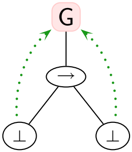
or

$$\forall (\beta) \forall (\alpha)\ \alpha \rightarrow \beta$$

  - We follow a lowermost-leftmost order

Computing $\bigwedge(g)$

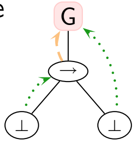
- We add a type quantification for all the nodes flexibly bound on g
  - But in which order?



$\forall(\alpha)\,\forall(\beta)\ \alpha \to \beta$
or
$\forall(\beta)\,\forall(\alpha)\ \alpha \to \beta$

  - We follow a lowermost-leftmost order

- Not sufficient: while



has type $\forall(\beta)\,\forall(\alpha)\ \alpha \to \beta$, a fresh instance of g has type $\forall(\alpha)\,\forall(\beta)\ \alpha \to \beta$ according to a leftmost order

- We sometimes need to insert reordering computations

Computing $\mathcal{T}(e)$

- One translation for each of the four instance operations

  Plus one new atomic operation RaiseMerge which is translated as $\alpha \triangleleft$

- Not very difficult (except for raising), but verbose, as the graphic and xML$^\mathsf{F}$ instance relations are very different

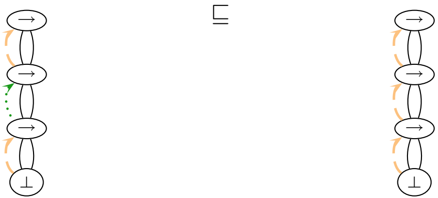
Computing $\mathcal{T}(e)$

- One translation for each of the four instance operations

  Plus one new atomic operation RaiseMerge which is translated as $\alpha \triangleleft$

- Not very difficult (except for raising), but verbose, as the graphic and xML$^{\mathsf{F}}$ instance relations are very different

- Some operations cannot be translated at all:



In xML$^{\mathsf{F}}$, $(\forall\,(\alpha \geqslant \bot \to \bot)\,\alpha \to \alpha) \to (\forall\,(\alpha \geqslant \bot \to \bot)\,\alpha \to \alpha) \not\leq$
$$((\bot \to \bot) \to (\bot \to \bot)) \to ((\bot \to \bot) \to (\bot \to \bot))$$

$\Rightarrow$ Not all presolutions can be translated

Correcteness of the translation

- Any presolution can be transformed into a translatable one
  - This can be done in a modular way
  - The translation preserves types modulo inert nodes
- Translatable presolutions are translated into well-typed $x\mathsf{ML}^{\mathsf{F}}$ terns

This ensures the type soundness of our type inference framework

- The translation can trivially be adapted to the modulo versions of ML^{F} (which also ensures their soundness)

Outline

Conclusion

xML^F is an internal language for ML^F with all the good metatheoretical properties

Perspectives:

- Understand the differences in expressivity between the instance relations of ML^F and xML^F
- Efficient generation of elaborated terms from presolutions

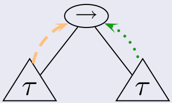
Coercions

- Annotated terms are not primitive, but syntactic sugar
  - $(a : \tau) \triangleq c_\tau\ a$
  - $\lambda(x : \tau)\ a \triangleq \lambda(x)$ let $x = (x : \tau)$ in a

- **Coercion functions**

  Primitives of the typing environment

  c_τ :

  - The domain of the arrow is frozen
  - The codomain can be freely instantiated

- in xMLF: $c_\tau \triangleq \Lambda(\alpha \geqslant \tau)\ \lambda(x : \tau)\ x[\alpha \triangleleft]$