xMLF, an explicit language for MLF

Who? Boris Yakobowski
Where? CNRS - University Paris 7
When? February 8, 2009
Outline

1. A brief summary of (graphic) MLF
2. A Church-style language for MLF
3. Translating graphic MLF into xMLF
4. Conclusion
Outline

1. A brief summary of (graphic) MLF
2. A Church-style language for MLF
3. Translating graphic MLF into xMLF
4. Conclusion
ML-like type inference + expressivity of System F second-order polymorphism
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-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:

\[
\begin{align*}
\text{id} & \triangleq \lambda(x) \, x & : & \forall \beta. \, \beta \rightarrow \beta \\
\text{choose} & \triangleq \lambda(x) \, \lambda(y) \, x & : & \forall \alpha. \, \alpha \rightarrow \alpha \rightarrow \alpha
\end{align*}
\]

\[
\text{choose id} : \begin{cases} 
(\forall \beta. \, \beta \rightarrow \beta) \rightarrow (\forall \beta. \, \beta \rightarrow \beta) & \alpha = \forall \beta. \, \beta \rightarrow \beta \\
\forall \gamma. \, (\gamma \rightarrow \gamma) \rightarrow (\gamma \rightarrow \gamma) & \alpha = \gamma \rightarrow \gamma
\end{cases}
\]

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 $\forall (\alpha \geq \tau) \tau'$:

Both $\tau$ and $\tau'$ can be instantiated inside $\forall (\alpha \geq \tau) \tau'$
All occurrences of $\alpha$ in $\tau'$ must pick the same instance of $\tau$

choose id : $\forall (\alpha \geq \forall \beta. \beta \to \beta) \alpha \to \alpha$

$\sqsubseteq (\forall \beta. \beta \to \beta) \to (\forall \beta. \beta \to \beta)$

or $\sqsubseteq \forall \gamma. (\gamma \to \gamma) \to (\gamma \to \gamma)$
MLF types: going beyond System F

To solve the problem of non-principality:

Flexible quantification

MLF types extend System F types with an instance-bounded quantification of the form \( \forall (\alpha \geq \tau) \tau' \):

- Both \( \tau \) and \( \tau' \) can be instantiated inside \( \forall (\alpha \geq \tau) \tau' \)
- All occurrences of \( \alpha \) in \( \tau' \) must pick the same instance of \( \tau \)

To permit type inference:

Rigid quantification

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

- \( \tau \) cannot (really) be instantiated inside \( \forall (\alpha = \tau) \tau' \)
- But \( \forall (\alpha = \tau) \alpha \rightarrow \alpha \) and \( \forall (\alpha = \tau) \forall (\alpha' = \tau) \alpha \rightarrow \alpha' \) are different as far as type inference is concerned
MLF 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**
- MLF 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 ML$^F$ types

The superposition of:

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

\[
\forall (\alpha \geq \bot) \forall (\gamma = \forall (\beta \geq \bot) \alpha \to \beta) \gamma \to \gamma
\]
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 kinds of binding edges, for flexible and rigid quantification

Sharing of nodes is important

\[ \forall (\alpha \geq \sigma_{id}) \alpha \rightarrow \alpha \]

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

\[ \forall (\alpha \geq \sigma_{id}) \forall (\beta \geq \sigma_{id}) \alpha \rightarrow \beta \]

Incorrect for $\lambda(x) \times$
Instance on graphic ML\textsuperscript{F} types

The instance relation $\sqsubseteq$

Four atomic operations on graphs:
Instance on graphic $\text{ML}_F$ types

The instance relation $\sqsubseteq$

Four atomic operations on graphs:

- **Grafting**: replacing a variable by a closed type (variable substitution)
Instance on graphic $\text{MLF}$ 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)
Instance on graphic $\text{ML}^F$ 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)
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 MLF 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**

- **Instantiation edges**
Type generalization

Type generalization is essential in MLF, just as in ML.

Gen nodes are used to promote types into type schemes, and to delimit generalization scopes.

\[ g : \forall \alpha. \alpha \rightarrow \alpha \]
Type generalization

Type generalization is essential in ML\textsuperscript{F}, just as in ML.

\textbf{Gen nodes} are used to promote types into type schemes, and to delimit generalization scopes.

\[ g : \forall \alpha. \alpha \rightarrow \alpha \]
\[ g' : \forall \beta. \beta \rightarrow \alpha \]

\(\alpha\) is free at the level of \(g'\)
Instantiation edges

Constrain a node to be an instance of a type scheme

Example:

\[ g \rightarrow \beta \perp \rightarrow n \rightarrow \alpha \]

\( e \) constrains \( n \) to be an instance of \( g \)
Instantiation edges

Constrain a node to be an instance of a type scheme

Example:

\[ g : \forall \beta. \beta \rightarrow \alpha \]
\[ n : \alpha \rightarrow \alpha \]

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:**

\[ g : \beta \rightarrow \alpha \]
\[ n : \alpha \rightarrow \alpha \]

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

e constrains \( n \) to be an instance of \( g \)
Typing constraint for an abstraction

\[ \lambda(x) \ a \xrightarrow{\sim} \top \]

\(\lambda(x)\ a\) can receive type \(\alpha \rightarrow \beta\), provided

- \(\alpha\) is the (common) type of all the occurrences of \(x\) in \(a\)
- \(\beta\) is an instance of the type of \(a\).
Typing constraint for an application

\[ a \ b \xrightarrow{\sim} G \]

\[ a \ b \] can receive type \( \beta \), provided there exists \( \alpha \) such that

- \( a \rightarrow \beta \) is an instance of the type of \( a \)
- \( \alpha \) is an instance of the type of \( b \)
Semantics of constraints

Presolutions

A presolution of a constraint \( \chi \) is an instance of \( \chi \) in which all the instantiation and unification edges are solved.

Presolutions retain the shape of the original constraint

Example: Constraint for \( \lambda(x) \cdot x \)
Outline

1. A brief summary of (graphic) MLF
2. A Church-style language for MLF
3. Translating graphic MLF into xMLF
4. Conclusion
An explicit language for $\text{MLF}$

Study subject reduction in $\text{MLF}$

- Type annotations are important inside terms
- But how to reduce $(e : \sigma)$?

How to use $\text{MLF}$ inside a typed compiler?

- $\text{MLF}$ types are more expressive than $\text{F}$ ones
- System $\text{F}$ cannot be used as a target language
  (prior work by Leijen, but not completely satisfactory)

Hence the need for a core, Church-style, language for $\text{MLF}$, $\text{xMLF}$
From System F to $\times\text{ML}^F$

$\times\text{ML}^F$ generalizes System F

Types: $\sigma ::= \bot \mid \forall (\alpha \geq \sigma) \sigma \mid \alpha \mid \sigma \rightarrow \sigma$

Rigid quantification is only needed for type inference, and is inlined in $\times\text{ML}^F$

Hence $\forall (\alpha = \sigma) \alpha \rightarrow \alpha$ becomes $\sigma \rightarrow \sigma$
From System F to $\times\text{MLF}$

$\times\text{MLF}$ generalizes System F

**Types:**
\[ \sigma ::= \bot \mid \forall (\alpha \geq \sigma) \sigma \mid \alpha \mid \sigma \rightarrow \sigma \]

Rigid quantification is only needed for type inference, and is inlined in $\times\text{MLF}$

Hence $\forall (\alpha = \sigma) \alpha \rightarrow \alpha$ becomes $\sigma \rightarrow \sigma$

**Terms:**
\[ a ::= x \mid \lambda(x: \sigma) a \mid a \ a \mid \text{let } x = a \text{ in } a \]
\[ \mid \Lambda(\alpha \geq \sigma) a \mid a[\varphi] \]

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

\[
\begin{align*}
T_{\text{APP}} \\
\Gamma \vdash a : \sigma & \quad \Gamma \vdash \varphi : \sigma \leq \sigma' \\
\hline
\Gamma \vdash a[\varphi] : \sigma' 
\end{align*}
\]
Type computations

Instance is explicitly witnessed through the use of type computations

\[ \varphi ::= \varepsilon \mid \varphi ; \varphi \mid \downarrow \sigma \mid \alpha \uparrow \mid \forall (\geq \varphi) \mid \forall (\alpha \geq) \varphi \mid \leq \mid \geq \]

<table>
<thead>
<tr>
<th>Inst-Reflex</th>
<th>Inst-Trans</th>
<th>Inst-Bot</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma \vdash \varepsilon : \sigma \leq \sigma ]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| \[ \Gamma \vdash \varphi_1 : \sigma_1 \leq \sigma_2 \quad \Gamma \vdash \varphi_2 : \sigma_2 \leq \sigma_3 \]
| \[ \Gamma \vdash \varphi_1 ; \varphi_2 : \sigma_1 \leq \sigma_3 \] |
| \[ \Gamma \vdash \triangleright \sigma : \bot \leq \sigma \] |

<table>
<thead>
<tr>
<th>Inst-Hyp</th>
<th>Inst-Inner</th>
</tr>
</thead>
</table>
| \[ \alpha \geq \sigma \in \Gamma \]
| \[ \Gamma \vdash \alpha \downarrow : \sigma \leq \alpha \] |
| \[ \Gamma \vdash \forall (\geq \varphi) : \forall (\alpha \geq \sigma_1) \sigma \leq \forall (\alpha \geq \sigma_2) \sigma \] |

<table>
<thead>
<tr>
<th>Inst-Outer</th>
<th>Inst-Quant-Elim</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma, \varphi : \alpha \geq \sigma \vdash \varphi : \sigma_1 \leq \sigma_2 ]</td>
<td></td>
</tr>
<tr>
<td>[ \Gamma \vdash \forall (\alpha \geq) \varphi : \forall (\alpha \geq \sigma) \sigma_1 \leq \forall (\alpha \geq \sigma) \sigma_2 ]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inst-Quant-Elim</th>
<th>Inst-Quant-Intro</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma \vdash &amp; : \forall (\alpha \geq \sigma) \sigma' \leq \sigma' { \alpha \leftarrow \sigma } ]</td>
<td></td>
</tr>
</tbody>
</table>
| \[ \alpha \notin \text{ftv}(\sigma) \]
| \[ \Gamma \vdash \triangleright \sigma : \sigma \leq \forall (\alpha \geq \bot) \sigma \] |
Type computations

Instance is explicitly witnessed through the use of type computations

\[ \varphi ::= \epsilon \mid \varphi; \varphi \mid \triangleright \sigma \mid \alpha \triangleleft \mid \forall (\geq \varphi) \mid \forall (\alpha \geq) \varphi \mid \& \mid \exists \]

\[
\begin{align*}
\text{Inst-Reflex} & \quad \Gamma \vdash \epsilon : \sigma \leq \sigma \\
\text{Inst-Trans} & \quad \Gamma \vdash \varphi_1 : \sigma_1 \leq \sigma_2 \quad \Gamma \vdash \varphi_2 : \sigma_2 \leq \sigma_3 \\
& \quad \Gamma \vdash \varphi_1; \varphi_2 : \sigma_1 \leq \sigma_3 \\
\text{Inst-Bot} & \quad \Gamma \vdash \triangleright \sigma : \bot \leq \sigma
\end{align*}
\]

\[
\begin{align*}
\text{Inst-Hyp} & \quad \alpha \geq \sigma \in \Gamma \\
& \quad \Gamma \vdash \alpha \triangleleft : \sigma \leq \alpha \\
\text{Inst-Inner} & \quad \Gamma \vdash \varphi : \sigma_1 \leq \sigma_2 \\
& \quad \Gamma \vdash \forall (\geq \varphi) : \forall (\alpha \geq \sigma_1) \sigma \leq \forall (\alpha \geq \sigma_2) \sigma
\end{align*}
\]

\[
\begin{align*}
\text{Inst-Outer} & \quad \Gamma, \varphi : \alpha \geq \sigma \vdash \varphi : \sigma_1 \leq \sigma_2 \\
& \quad \Gamma \vdash \forall (\alpha \geq) \varphi : \forall (\alpha \geq \sigma) \sigma_1 \leq \forall (\alpha \geq \sigma) \sigma_2
\end{align*}
\]

\[
\begin{align*}
\text{Inst-Quant-Elim} & \quad \Gamma \vdash \& : \forall (\alpha \geq \sigma) \sigma' \leq \sigma' \{\alpha \leftarrow \sigma\} \\
\text{Inst-Quant-Intro} & \quad \alpha \notin \text{ftv}(\sigma) \\
& \quad \Gamma \vdash \& : \sigma \leq \forall (\alpha \geq \bot) \sigma
\end{align*}
\]
# Type computations

Instance is **explicitely witnessed** through the use of type computations

\[
\varphi ::= \varepsilon \mid \varphi; \varphi \mid \triangleright \sigma \mid \alpha \triangleleft \mid \forall (\geq \varphi) \mid \forall (\alpha \geq \sigma) \varphi \mid \& \mid \varnothing
\]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inst-Reflex</td>
<td>( \Gamma \vdash \varepsilon : \sigma \leq \sigma )</td>
<td>( \Gamma \vdash \varepsilon : \sigma \leq \sigma )</td>
</tr>
<tr>
<td>Inst-Trans</td>
<td>( \Gamma \vdash \varphi_1 : \sigma_1 \leq \sigma_2 )</td>
<td>( \Gamma \vdash \varphi_2 : \sigma_2 \leq \sigma_3 )</td>
</tr>
<tr>
<td>Inst-Bot</td>
<td>( \Gamma \vdash \triangleright \sigma : \bot \leq \sigma )</td>
<td></td>
</tr>
<tr>
<td>Inst-Hyp</td>
<td>( \alpha \geq \sigma \in \Gamma )</td>
<td>( \Gamma \vdash \alpha \triangleleft : \sigma \leq \alpha )</td>
</tr>
<tr>
<td>Inst-Inner</td>
<td>( \Gamma \vdash \varphi : \sigma_1 \leq \sigma_2 )</td>
<td>( \Gamma \vdash \forall (\geq \varphi) : \forall (\alpha \geq \sigma_1) \sigma \leq \forall (\alpha \geq \sigma_2) \sigma )</td>
</tr>
<tr>
<td>Inst-Outer</td>
<td>( \Gamma, \varphi : \alpha \geq \sigma \vdash \varphi : \sigma_1 \leq \sigma_2 )</td>
<td>( \Gamma \vdash \forall (\alpha \geq \sigma) \varphi : \forall (\alpha \geq \sigma) \sigma_1 \leq \forall (\alpha \geq \sigma) \sigma_2 )</td>
</tr>
<tr>
<td>Inst-Quant-Elim</td>
<td>( \Gamma \vdash &amp; : \forall (\alpha \geq \sigma) \sigma' \leq \sigma'{\alpha \leftarrow \sigma} )</td>
<td></td>
</tr>
<tr>
<td>Inst-Quant-Intro</td>
<td>( \alpha \notin \text{ftv}(\sigma) )</td>
<td>( \Gamma \vdash \varnothing : \sigma \leq \forall (\alpha \geq \bot) \sigma )</td>
</tr>
</tbody>
</table>
**Type computations**

Instance is explicitly witnessed through the use of type computations

\[ \varphi ::= \varepsilon \mid \varphi; \varphi \mid \triangleright \sigma \mid \alpha \triangleleft \mid \forall (\geq \varphi) \mid \forall (\alpha \geq) \varphi \mid \& \mid \checkmark \]

**Inst-Reflex**

\[ \Gamma \vdash \varepsilon : \sigma \leq \sigma \]

**Inst-Trans**

\[ \Gamma \vdash \varphi_1 : \sigma_1 \leq \sigma_2 \quad \Gamma \vdash \varphi_2 : \sigma_2 \leq \sigma_3 \]

\[ \Gamma \vdash \varphi_1; \varphi_2 : \sigma_1 \leq \sigma_3 \]

**Inst-Bot**

\[ \Gamma \vdash \triangleright \sigma : \bot \leq \sigma \]

**Inst-Hyp**

\[ \alpha \geq \sigma \in \Gamma \]

\[ \Gamma \vdash \alpha \triangleleft : \sigma \leq \alpha \]

**Inst-Inner**

\[ \Gamma \vdash \varphi : \sigma_1 \leq \sigma_2 \]

\[ \Gamma \vdash \forall (\geq \varphi) : \forall (\alpha \geq \sigma_1) \sigma \leq \forall (\alpha \geq \sigma_2) \sigma \]

**Inst-Outer**

\[ \Gamma, \varphi : \alpha \geq \sigma \vdash \varphi : \sigma_1 \leq \sigma_2 \]

\[ \Gamma \vdash \forall (\alpha \geq) \varphi : \forall (\alpha \geq \sigma) \sigma_1 \leq \forall (\alpha \geq \sigma) \sigma_2 \]

**Inst-Quant-Elim**

\[ \Gamma \vdash \& : \forall (\alpha \geq \sigma) \sigma' \leq \sigma'\{\alpha \leftarrow \sigma\} \]

**Inst-Quant-Intro**

\[ \alpha \notin \text{ftv}(\sigma) \]

\[ \Gamma \vdash \triangledown : \sigma \leq \forall (\alpha \geq \bot) \sigma \]
Type computations

Instance is explicitly witnessed through the use of type computations

\[ \varphi ::= \varepsilon \mid \varphi; \varphi \mid \triangleright \sigma \mid \alpha \triangleleft \mid \forall (\triangleright \varphi) \mid \forall (\triangleright \alpha) \varphi \mid \& \mid \bot \]

### Inst-Reflex

\[ \Gamma \vdash \varepsilon : \sigma \leq \sigma \]

### Inst-Trans

\[ \Gamma \vdash \varphi_1 : \sigma_1 \leq \sigma_2 \]
\[ \Gamma \vdash \varphi_2 : \sigma_2 \leq \sigma_3 \]
\[ \Gamma \vdash \varphi_1 \triangleright \varphi_2 : \sigma_1 \leq \sigma_3 \]

### Inst-Bot

\[ \Gamma \vdash \triangleright \sigma : \bot \leq \sigma \]

### Inst-Hyp

\[ \alpha \triangleright \sigma \in \Gamma \]
\[ \Gamma \vdash \alpha \triangleleft : \sigma \leq \alpha \]

### Inst-Inner

\[ \Gamma \vdash \varphi : \sigma_1 \leq \sigma_2 \]
\[ \Gamma \vdash \forall (\triangleright \varphi) : \forall (\triangleright \alpha \triangleright \sigma_1) \sigma \leq \forall (\triangleright \alpha \triangleright \sigma_2) \sigma \]

### Inst-Outer

\[ \Gamma, \varphi : \alpha \triangleright \sigma \vdash \varphi : \sigma_1 \leq \sigma_2 \]
\[ \Gamma \vdash \forall (\triangleright \alpha) \varphi : \forall (\triangleright \alpha \triangleright \sigma) \sigma_1 \leq \forall (\triangleright \alpha \triangleright \sigma) \sigma_2 \]

### Inst-Quant-Elim

\[ \Gamma \vdash \& : \forall (\triangleright \alpha \triangleright \sigma) \sigma' \leq \sigma' \{ \alpha \leftarrow \sigma \} \]

### Inst-Quant-Intro

\[ \alpha \notin \text{ftv}(\sigma) \]
\[ \Gamma \vdash \triangleright \sigma : \sigma \leq \forall (\triangleright \alpha \triangleright \bot) \sigma \]
Example: back to choose id

\[
\begin{align*}
\text{choose} &\triangleq \Lambda(\alpha \geq \bot) \ \lambda(x : \alpha) \ \lambda(y : \alpha) \ x : \ \forall (\alpha \geq \bot) \ \alpha \to \alpha \to \alpha \\
\text{id} &\triangleq \Lambda(\beta \geq \bot) \ \lambda(x : \beta) \ x \quad : \quad \forall (\beta \geq \bot) \ \beta \to \beta
\end{align*}
\]

To make choose \text{id} \textbf{well-typed}, we must choose a type into which \(\alpha\) must be instantiated.
Example: back to choose id

\[
\text{choose } \triangleq \Lambda(\alpha \geq \bot) \lambda(x : \alpha) \lambda(y : \alpha) x : \forall (\alpha \geq \bot) \alpha \to \alpha \to \alpha
\]

\[
\text{id } \triangleq \Lambda(\beta \geq \bot) \lambda(x : \beta) x : \forall (\beta \geq \bot) \beta \to \beta
\]

To make choose id well-typed, we must choose a type into which \(\alpha\) must be instantiated

\[
e \triangleq \Lambda(\gamma \geq \sigma_{id}) (\text{choose}[\forall (\geq \triangleright \gamma); \&]) (\text{id}[\gamma \triangleright]) : \forall (\gamma \geq \sigma_{id}) \gamma \to \gamma
\]

\[
\Gamma \vdash \triangleright \gamma : \bot \leq \gamma
\]

\[
\Gamma \vdash \forall (\geq \triangleright \gamma) : \forall (\alpha \geq \bot) \alpha \to \alpha \to \alpha \leq \forall (\alpha \geq \gamma) \alpha \to \alpha \to \alpha \text{ INNER}
\]

\[
\Gamma \vdash \& : \forall (\alpha \geq \gamma) \alpha \to \alpha \to \alpha \leq \gamma \to \gamma \to \gamma \text{ QUANT-ELIM}
\]

\[
\Gamma \vdash \forall (\geq \triangleright \gamma); \& : \forall (\alpha \geq \bot) \alpha \to \alpha \to \alpha \leq \gamma \to \gamma \to \gamma \text{ TRANS}
\]
Example: back to choose id

\[
\text{choose} \triangleq \Lambda(\alpha \geq \bot) \lambda(x : \alpha) \lambda(y : \alpha) x : \forall (\alpha \geq \bot) \alpha \rightarrow \alpha \rightarrow \alpha
\]
\[
\text{id} \triangleq \Lambda(\beta \geq \bot) \lambda(x : \beta) x : \forall (\beta \geq \bot) \beta \rightarrow \beta
\]

To make choose id well-typed, we must choose a type into which \(\alpha\) must be instantiated

\[
e \triangleq \Lambda(\gamma \geq \sigma_{id}) \left( \text{choose}[\forall (\geq \triangleright \gamma); \&]) \right) \left( \text{id}[\gamma \triangleleft] \right) : \forall (\gamma \geq \sigma_{id}) \gamma \rightarrow \gamma
\]

We can recover the other System F types just by instantiation

\[
\left\{ \begin{array}{ll}
e[\&] & : \sigma_{id} \rightarrow \sigma_{id} \\
\& (\forall (\delta \geq) (\forall (\geq \triangleright \delta); \&); \&]) & : \forall (\delta \geq \bot) (\delta \rightarrow \delta) \rightarrow (\delta \rightarrow \delta)
\end{array} \right.
\]
Reducing expressions

Usual $\beta$-reduction

\[(\lambda(x: \tau) \ a_1) \ a_2 \quad \rightarrow \quad a_1\{x \leftarrow a_2\}\quad \quad (\beta)\]

\[\text{let } x = a_2 \text{ in } a_1 \quad \rightarrow \quad a_1\{x \leftarrow a_2\}\quad \quad (\beta_{\text{LET}})\]
Reducing expressions

Usual $\beta$-reduction

6 specific rules to reduce type applications

\[
\begin{align*}
(\lambda(x : \tau) \: a_1) \: a_2 & \longrightarrow a_1\{x \leftarrow a_2\} \\
\text{let } x = a_2 \text{ in } a_1 & \longrightarrow a_1\{x \leftarrow a_2\} \\

a[\varepsilon] & \longrightarrow a \\

a[\varphi; \varphi'] & \longrightarrow a[\varphi][\varphi'] \\

a[\emptyset] & \longrightarrow \Lambda(\alpha \geq \bot) \: a \\
& \text{if } \alpha \notin \text{ftv}(a)
\end{align*}
\]

\[
\begin{align*}
(\Lambda(\alpha \geq \tau) \: a)[\forall (\alpha \geq) \: \varphi] & \longrightarrow \Lambda(\alpha \geq \tau) \: (a[\varphi]) \\
(\Lambda(\alpha \geq \tau) \: a)[\forall (\geq \varphi)] & \longrightarrow \Lambda(\alpha \geq \tau[\varphi]) \: a\{\alpha \leftarrow \varphi; \alpha \leftarrow\} \\
(\Lambda(\alpha \geq \tau) \: a)[\land] & \longrightarrow a\{\alpha \leftarrow \varepsilon\}\{\alpha \leftarrow \tau\}
\end{align*}
\]
Reducing expressions

Usual $\beta$-reduction

6 specific rules to reduce type applications

Context rule

$$E ::= \{\cdot\} \mid E[\varphi] \mid \lambda(x:\tau) \, E \mid \Lambda(\alpha \geq \tau) \, E$$

$$\mid E \, a \mid a \, E \mid \text{let } x = E \text{ in } a \mid \text{let } x = a \text{ in } E$$

\[
\begin{align*}
(\lambda(x:\tau) \, a_1) \, a_2 & \rightarrow a_1\{x \leftarrow a_2\} & \text{(}\beta\text{)} \\
\text{let } x = a_2 \text{ in } a_1 & \rightarrow a_1\{x \leftarrow a_2\} & \text{(}\beta\text{}_{\text{LET}}\text{)} \\
\end{align*}
\]

\[
\begin{align*}
a[\varepsilon] & \rightarrow a & \text{REFLEX} \\
a[\varphi; \varphi'] & \rightarrow a[\varphi][\varphi'] & \text{TRANS} \\
a[\forall] & \rightarrow \Lambda(\alpha \geq \bot) \, a & \text{QUANT-INTRO} \\
\text{if } \alpha \notin \text{ftv}(a) & \\
\end{align*}
\]

\[
\begin{align*}
(\Lambda(\alpha \geq \tau) \, a)[\forall (\alpha \geq \varphi)] & \rightarrow \Lambda(\alpha \geq \tau) \, (a[\varphi]) & \text{OUTER} \\
(\Lambda(\alpha \geq \tau) \, a)[\forall (\varphi)] & \rightarrow \Lambda(\alpha \geq \tau[\varphi]) \, a\{\alpha \leftarrow \varphi\} & \text{INNER} \\
(\Lambda(\alpha \geq \tau) \, a)[\&] & \rightarrow a\{\alpha \leftarrow \varepsilon\}\{\alpha \leftarrow \tau\} & \text{QUANT-ELIM} \\
\end{align*}
\]

\[
\begin{align*}
E\{a\} & \rightarrow E\{a'\} & \text{CONTEXT} \\
\text{if } a \rightarrow a' & \\
\end{align*}
\]
Rules **INNER** and **QUANT-ELIM**

\[(\Lambda(\alpha \geq \tau) \ a)[\forall (\geq \varphi)] \rightarrow \Lambda(\alpha \geq \tau[\varphi]) \ a \ ?\]

\[(\Lambda(\alpha \geq \tau) \ a)[\&] \rightarrow \ a\{\alpha \leftarrow \tau\} \ ?\]

This is incorrect: after the reduction, the computations \(\alpha \triangleleft\) inside \(a\) make incorrect assumptions on the bound of \(\alpha\)
Rules **Inner** and **Quant-Elim**

\[(\Lambda(\alpha \geq \tau) \ a)[\forall (\geq \varphi)] \quad \rightarrow \quad \Lambda(\alpha \geq \tau[\varphi]) \ a\{\alpha \leftarrow \varphi; \alpha \lessdot\}\] 

\[(\Lambda(\alpha \geq \tau) \ a)[\&] \quad \rightarrow \quad a\{\alpha \leftarrow \varepsilon\}\{\alpha \leftarrow \tau\}\]

This is **incorrect**: after the reduction, the computations \(\alpha \lessdot\) inside \(a\) make incorrect assumptions on the bound of \(\alpha\)

We change those computations:

- For **Inner**, \(\alpha \lessdot\) assumed that the bound of \(\alpha\) was \(\tau\), while it is \(\tau[\varphi]\)
- For **Quant-Elim**, \(\alpha\) is now \(\tau\), the computations \(\alpha \lessdot\) are vacuous
Example of reductions

choose id:

\[ \Lambda(\gamma \geq \sigma_{id}) (((\Lambda(\alpha \geq \bot) \lambda(x : \alpha) \lambda(y : \alpha) x)[\forall (\geq \triangleright \gamma); \&]) (id[\gamma \triangleleft])) \]
\[ \rightarrow \Lambda(\gamma \geq \sigma_{id}) (((\Lambda(\alpha \geq \gamma) \lambda(x : \alpha) \lambda(y : \alpha) x)[\&]) (id[\gamma \triangleleft])) \]
\[ \rightarrow \Lambda(\gamma \geq \sigma_{id}) (\lambda(x : \gamma) \lambda(y : \gamma) x)) (id[\gamma \triangleleft])) \]
\[ \rightarrow \Lambda(\gamma \geq \sigma_{id}) \lambda(y : \gamma) (id[\gamma \triangleleft])) \]

(choose id)[\&]:

\[ ((\Lambda(\gamma \geq \sigma_{id}) \lambda(y : \gamma) (id[\gamma \triangleleft])))[\&] \]
\[ \rightarrow \lambda(x : \sigma_{id}) (id[\epsilon]) \]
\[ \rightarrow \lambda(x : \sigma_{id}) \ id \]
Example of reductions

choose id:

\[ \Lambda(\gamma \geq \sigma_{id}) \ ((\Lambda(\alpha \geq \bot) \ \lambda(x : \alpha) \ \lambda(y : \alpha) \ x)[\forall (\geq \triangleright \gamma); \&]) \ (id[\gamma \triangleleft]) \]
\[ \rightarrow \ \Lambda(\gamma \geq \sigma_{id}) \ ((\Lambda(\alpha \geq \gamma) \ \lambda(x : \alpha) \ \lambda(y : \alpha) \ x)[\&]) \ (id[\gamma \triangleleft]) \]
\[ \rightarrow \ \Lambda(\gamma \geq \sigma_{id}) \ (\lambda(x : \gamma) \ \lambda(y : \gamma) \ x)) \ (id[\gamma \triangleleft]) \]
\[ \rightarrow \ \Lambda(\gamma \geq \sigma_{id}) \ \lambda(y : \gamma) \ (id[\gamma \triangleleft]) \]

(choose id)[&]:

\[ (\Lambda(\gamma \geq \sigma_{id}) \ \lambda(y : \gamma) \ (id[\gamma \triangleleft]))[\&] \]
\[ \rightarrow \ \lambda(x : \sigma_{id}) \ (id[\epsilon]) \]
\[ \rightarrow \ \lambda(x : \sigma_{id}) \ id \]

System F like type application

\[ [\tau] \triangleq [\forall (\geq \triangleright \tau); \&] \]
\[ (\Lambda(\alpha) \ a)[\tau] = (\Lambda(\alpha \geq \bot) \ a)[\forall (\geq \triangleright \tau); \&] \]
\[ \rightarrow (\Lambda(\alpha \geq \tau) \ a)[\&] \]
\[ \rightarrow a\{\alpha \leftarrow \tau\} \]

⇒ Exactly as in System F
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:

\[ e \triangleq (\Lambda(\alpha \geq \forall \gamma \gamma) \ ((\Lambda(\beta \geq \text{int}) x)[\forall (\geq \alpha \triangle)]) )[\forall (\geq \&)] \]

Ill-typed because the computation \( \alpha \triangle \) is applied to \text{int}, while \( \alpha \) is supposed to be \( \forall \gamma \gamma \)

\[ e \rightarrow (\Lambda(\alpha \geq \forall \gamma \gamma) \Lambda(\beta \geq \alpha) x)[\forall (\geq \&)] \]
\[ \rightarrow \Lambda(\alpha \geq \bot) \Lambda(\beta \geq \alpha) x \]

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

\[ e \rightarrow \Lambda(\alpha \geq \bot) ((\Lambda(\beta \geq \text{int}) x)[\forall (\geq \&; \alpha \triangle)]) \]

(Reducing the outermost type application first)
Correctness

- **Subject reduction**, under any context (including under $\lambda$ and $\Lambda$)

- Progress for **call-by-value**, with or without the value restriction, and for **call-by-name**

  First time that MLF is proven sound for call-by-name
Correctness

Subject reduction, under any context (including under $\lambda$ and $\Lambda$)

Progress for call-by-value, with or without the value restriction, and for call-by-name

First time that $\text{MLF}$ is proven sound for call-by-name

Mechanized proof?

- almost completed on a previous version of the system, in which $\varepsilon$, $\triangleright \tau$ 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 $\text{ML}^F$, $\lambda(x)\ x$ can receive the type

$$\tau \triangleq \forall (\alpha \geq \bot) \forall (\beta \geq \alpha) \beta \rightarrow \alpha$$

which is equivalent to $\forall (\alpha \geq \bot) \alpha \rightarrow \alpha$

In $x\text{ML}^F$, $\tau \leq \tau'' \rightarrow \tau'$, for any $\tau'$ and $\tau''$ such that $\vdash \varphi: \tau' \leq \tau''$

(as witnessed by $\forall (\geq \triangleright \tau); \&; \forall (\geq \varphi); \&$)
Alias bounds

In the syntactic presentations of MLF, $\lambda(x) \ x$ can receive the type

$$\tau \triangleq \forall (\alpha \geq \bot) \forall (\beta \geq \alpha) \beta \rightarrow \alpha$$

which is equivalent to $\forall (\alpha \geq \bot) \alpha \rightarrow \alpha$

In $xMLF$, $\tau \leq \tau'' \rightarrow \tau'$, for any $\tau'$ and $\tau''$ such that $\vdash \varphi : \tau' \leq \tau''$ (as witnessed by $\forall (\geq \varphi$); $\&$; $\forall (\geq \varphi$); $\&$)

Those types are in general incorrect for the identity!

Thankfully, $\lambda(x) \ x$ cannot receive type $\tau$ in $xMLF$.

Still, $xMLF$ types are (strictly) more expressive than the usual syntactic MLF types
Outline

1. A brief summary of (graphic) MLF
2. A Church-style language for MLF
3. Translating graphic MLF into xMLF
4. Conclusion
From presolutions to $x\text{MLF}^F$ terms

$\text{MLF}^F$ presolutions can be algorithmically translated into $x\text{MLF}^F$ terms
From presolutions to $x\text{MLF}$ terms

- $\text{MLF}$ presolutions can be algorithmically translated into $x\text{MLF}$ terms.
  - Nodes flexibly bound on gen nodes are translated into $x\text{MLF}$ 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 $x\text{MLF}$.
  - Some differences between the instance relations of $\text{MLF}$ and $x\text{MLF}$.
From presolutions to $x\text{ML}^F$ terms: example

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

Here, $K : \forall (\alpha) \alpha \rightarrow \sigma_{id} \rightarrow \alpha$
From presolutions to $xML^F$ terms: example

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

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

$$\Lambda(\alpha) \, \lambda(x : \alpha) \, (\Lambda(\beta) \, \lambda(y : \beta) \, x)$$

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

$$\sigma_{id} \rightarrow \alpha$$
From presolutions to $\text{XML}^F$ terms: example

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

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

\[
\begin{align*}
\Lambda(\alpha) \lambda(x : \alpha) \ (\Lambda(\beta) \lambda(y : \beta) x) \quad &\quad \left[\forall (\geq \sigma_{id}); \&\right] \\
n &\quad \forall (\beta) \beta \rightarrow \alpha \\
&\quad \sigma_{id} \rightarrow \alpha
\end{align*}
\]
Gen nodes and $xML^F$ terms

**Example:** $id \ id$

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 $\lambda$-terms

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

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

\[
[a_1 \ a_2] = \bigwedge(g) ([a_1][T(e_1)]) ([a_2][T(e_2)])
\quad (3)
\]

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

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

\[
\forall (\alpha) \forall (\beta) \alpha \rightarrow \beta
\]
\[\text{or}\]
\[
\forall (\beta) \forall (\alpha) \alpha \rightarrow \beta
\]

- 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?

We follow a lowermost-leftmost order

Not sufficient: while $G \rightarrow \bot \bot$ has type $\forall (\beta) \forall (\alpha) \alpha \rightarrow \beta$,

a fresh instance of $g$ has type $\forall (\alpha) \forall (\beta) \alpha \rightarrow \beta$ 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 $\text{RaiseMerge}$ which is translated as $\alpha \downarrow$
- Not very difficult (except for raising), but verbose, as the graphic and $\times\text{MLF}$ instance relations are very different
Computing $T(e)$

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 $xML^F$ instance relations are very different

Some operations cannot be translated at all:

\[ \begin{array}{c}
\rightarrow \\
\rightarrow \\
\square \\
\rightarrow \\
\rightarrow \\
\perp
\end{array} \quad \begin{array}{c}
\subseteq \\
\rightarrow \\
\rightarrow \\
\perp
\end{array} \quad \begin{array}{c}
\rightarrow \\
\rightarrow \\
\perp
\end{array} \]

In $xML^F$, $(\forall (\alpha \geq \perp \rightarrow \perp) \alpha \rightarrow \alpha) \rightarrow (\forall (\alpha \geq \perp \rightarrow \perp) \alpha \rightarrow \alpha) \not\leq ((\perp \rightarrow \perp) \rightarrow (\perp \rightarrow \perp)) \rightarrow ((\perp \rightarrow \perp) \rightarrow (\perp \rightarrow \perp))$

$\Rightarrow$ 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 $x\text{ML}^F$ terms.
  - This ensures the type soundness of our type inference framework.

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

1. A brief summary of (graphic) MLF
2. A Church-style language for MLF
3. Translating graphic MLF into xMLF
4. Conclusion
Conclusion

\(x\text{MLF}\) is an internal language for \(\text{MLF}\) with all the good metatheoretical properties

**Perspectives:**

- Understand the *differences in expressivity* between the instance relations of \(\text{MLF}\) and \(x\text{MLF}\)

- **Efficient** generation of elaborated terms from presolutions
Coercions

Annotated terms are not primitive, but syntaxic sugar

\[(a : \tau) \triangleq c_{\tau} \ a\]
\[\lambda(x : \tau) \ a \triangleq \lambda(x) \ \text{let} \ x = (x : \tau) \ \text{in} \ a\]

Coercion functions

Primitives of the typing environment

\[c_{\tau} : \tau \rightarrow \tau\]

The domain of the arrow is frozen

The codomain can be freely instantiated

\[\text{in } \text{xMLF:} \quad c_{\tau} \triangleq \Lambda(\alpha \geq \tau) \ \lambda(x : \tau) \ x[\alpha <]\]