A (quick) tour of MLF

(Graphic) Types

(Type) Constraints

Solving constraints

Type inference

Type Soundness
A Fully Graphical Presentation of MLF

Didier Rémy
&
Boris Yakobowski

Didier Le Botlan

INRIA
Rocquencourt
A (quick) tour of ML²

(Graphic) Types

(Type) Constraints

Solving constraints

Type inference

Type Soundness
The original $\text{ML}^F$

- It is *intuitively* simple, but
- Its *purely syntactic* presentation is *technically* involved.

**Is it the right definition?**
(It is sound, indeed, but where there other better choices?)

No, it is not quite right!

We now have the right definition, **twice**:

- “Semantically”, in *Recasting MLF* (work with Didier Le Botlan)
- Graphically, in this work
A new fully graphical presentation of Full ML^F

We build on previous work

A Graphical presentation of ML^F types (TLDI 06)

Types and the instance relation are either well-known or simple operations on graphs. Allows for an efficient unification algorithm.

We

- enrich types with type constraints.
- solve type constraints by reducing them to unification problems.
- express type inference as typing constraints
- obtain a type-inference algorithm about as efficient as the one for ML
- show type soundness by reasoning on graphical typing constraints.
The key to $\text{ML}^F$

let choose $= \lambda(x) \lambda(y) \text{if } true \text{ then } x \text{ else } y : \forall \alpha \cdot \alpha \to \alpha \to \alpha$

let $id = \lambda(z) \ z : \forall \alpha \cdot \alpha \to \alpha$

choose $(\lambda(x) \ x) :$
let choose = \( \lambda(x) \lambda(y) \) if true then \( x \) else \( y \) : \( \forall \alpha \cdot \alpha \rightarrow \alpha \rightarrow \alpha \)

let \( id = \lambda(z) z \) : \( \forall \alpha \cdot \alpha \rightarrow \alpha \)

choose \((\lambda(x) x)\) : \[
\begin{cases}
\forall \alpha \cdot (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha) \\
(\forall \alpha \cdot \alpha \rightarrow \alpha) \rightarrow (\forall \alpha \cdot \alpha \rightarrow \alpha)
\end{cases}
\]
The key to ML$^F$

let choose $= \lambda(x) \lambda(y) \textbf{if} \; \text{true} \; \textbf{then} \; x \; \textbf{else} \; y : \forall \alpha \cdot \alpha \rightarrow \alpha \rightarrow \alpha$

let $id = \lambda(z) \; z : \forall \alpha \cdot \alpha \rightarrow \alpha$

choose $(\lambda(x) \; x) : \left\{ \begin{array}{l}
\forall \alpha \cdot (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha) \\
(\forall \alpha \cdot \alpha \rightarrow \alpha) \rightarrow (\forall \alpha \cdot \alpha \rightarrow \alpha) \end{array} \right\}$ No better choice in $F$
let choose = $\lambda(x) \lambda(y)$ if true then $x$ else $y : \forall \alpha \cdot \alpha \rightarrow \alpha \rightarrow \alpha$

let id = $\lambda(z) z : \forall \alpha \cdot \alpha \rightarrow \alpha$

choose $\left(\lambda(x) x\right) : \begin{cases} \forall \alpha \cdot (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha) \\ (\forall \alpha \cdot \alpha \rightarrow \alpha) \rightarrow (\forall \alpha \cdot \alpha \rightarrow \alpha) \end{cases}$ \hspace{1cm} \text{No better choice in } F$

: $\forall (\beta > \forall (\alpha) \alpha \rightarrow \alpha) \beta \rightarrow \beta \ \text{in } ML^F$
The key to ML^F

let choose = \lambda(x) \lambda(y) if true then x else y : \forall \alpha \cdot \alpha -\to \alpha -\to \alpha

let id = \lambda(z) z : \forall \alpha \cdot \alpha -\to \alpha

choose (\lambda(x) x) : \left\{ \begin{array}{l}
\forall \alpha \cdot (\alpha -\to \alpha) -\to (\alpha -\to \alpha) \\
(\forall \alpha \cdot \alpha -\to \alpha) -\to (\forall \alpha \cdot \alpha -\to \alpha)
\end{array} \right\}

: \forall (\beta > \forall (\alpha) \alpha -\to \alpha) \beta -\to \beta \quad \text{in ML}^F

\leq \left\{ \begin{array}{l}
\forall (\beta = \forall (\alpha) \alpha -\to \alpha) \beta -\to \beta \\
\forall (\alpha) \forall (\beta = \alpha -\to \alpha) \beta -\to \beta
\end{array} \right\}
The key to $\text{ML}^F$

let $\text{choose} = \lambda(x) \lambda(y) \text{ if } true \text{ then } x \text{ else } y : \forall \alpha \cdot \alpha \rightarrow \alpha \rightarrow \alpha$

let $id = \lambda(z) z : \forall \alpha \cdot \alpha \rightarrow \alpha$

choose $(\lambda(x) x) : \left\{ \begin{array}{l} \forall \alpha \cdot (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha) \\ (\forall \alpha \cdot \alpha \rightarrow \alpha) \rightarrow (\forall \alpha \cdot \alpha \rightarrow \alpha) \end{array} \right\}$ No better choice in $\text{F}$

: $\forall(\beta > \forall(\alpha) \alpha \rightarrow \alpha) \beta \rightarrow \beta$ in $\text{ML}^F$

$\leq \left\{ \begin{array}{l} \forall(\beta = \forall(\alpha) \alpha \rightarrow \alpha) \beta \rightarrow \beta \\ \forall(\alpha) \forall(\beta = \alpha \rightarrow \alpha) \beta \rightarrow \beta \end{array} \right\}$

But

$\lambda(x) x x$ : ill-typed, as we do not guess polymorphism!

$\lambda(x : \forall(\alpha) \alpha \rightarrow \alpha) x x$ : $\forall(\beta = \forall(\alpha) \alpha \rightarrow \alpha) \beta \rightarrow \beta$
The key to $\text{MLF}^F$

let choose $= \lambda(x) \lambda(y) \text{if } \text{true} \text{ then } x \text{ else } y : \forall \alpha \cdot \alpha \rightarrow \alpha \rightarrow \alpha$

let $id = \lambda(z) z : \forall \alpha \cdot \alpha \rightarrow \alpha$

choose $\lambda(x)$ $x$ : \[
\begin{cases} 
\forall \alpha \cdot (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha) \\
(\forall \alpha \cdot \alpha \rightarrow \alpha) \rightarrow (\forall \alpha \cdot \alpha \rightarrow \alpha)
\end{cases}
\]  
No better choice in $F$

Function parameters that are used polymorphically and only those need an annotation.

But

$\lambda(x) \ x \ x$ : ill-typed, as we do not guess polymorphism!

$\lambda(x : \forall(\alpha) \alpha \rightarrow \alpha) \ x \ x$ : $\forall(\beta = \forall(\alpha) \alpha \rightarrow \alpha) \beta \rightarrow \beta$
Graphical Types

∀ \left( \alpha \geq \forall(\beta) \beta \rightarrow \beta \right) \alpha \rightarrow (\alpha \rightarrow \alpha)
∀(\(\alpha \geq \forall(\beta) \beta \rightarrow \beta\)) \alpha \rightarrow (\alpha \rightarrow \alpha)
Graphical Types

∀ \left( \alpha \geq \forall (\beta) \beta \rightarrow \beta \right) \alpha \rightarrow (\alpha \rightarrow \alpha)
∀(β) ∀ \left( \alpha \geq \beta \rightarrow \beta \right) \alpha \rightarrow \alpha \rightarrow \alpha
∀ \left( \alpha \geq \forall \beta \beta \rightarrow \beta \right) \alpha \rightarrow (\alpha \rightarrow \alpha)
Graphical Types

Binding all nodes allows for a more regular representation

\[ \forall \left( \alpha \geq \forall (\beta) \beta \rightarrow \beta \right) \forall \left( \gamma \geq \alpha \rightarrow \alpha \right) \alpha \rightarrow \gamma \]
Superposition of a term-dag
(first-order terms sharing suffixes)
Graphical Types

and a binding tree structure

(± well-formedness conditions between both)
Graphical Types
Instance relation

Grafting
Graphical Types

Instance relation
Raising
Graphical Types

Instance relation

Raising
Graphical Types

Instance relation
Merging
Instance relation

Weakening

Changes permissions
Instance relation

Weakening

Changes permissions
Instance relation

Weakening

Changes permissions

\[
\forall \left( \alpha = \forall(\beta) \beta \rightarrow \beta \right) \forall \left( \gamma \geq \alpha \rightarrow \alpha \right) \alpha \rightarrow \gamma
\]
The interior of a node $n$ is the set of nodes, said inner $n$, that are transitivity bound at $n$. 

![Diagram showing the interior of a node with arrows indicating transitivity bound relationships.]
It can be transformed *locally*, as long as the interface (structure edges crossing the frontier) is maintained.
It can be transformed *locally*, as long as the interface (structure edges crossing the frontier) is maintained.
Constraints

They are graphic types extended with...

1) Existential nodes and unification edges
They are graphic types extended with...  

2) Type schemes and instantiation edges

Use sorts Scheme and Type to constraint formation of edges
Semantics of constraints

General picture

- Constraints are given a meaning as a set of types that are solutions of the constraints.

  (Hence two constraints with the same meaning may have completely different shapes, e.g. two unsolvable constraints are equivalent)

- Constraints are also types. As such, they can be instantiated along $\sqsubseteq$. 
Semantics of constraints

General picture

- Constraints are given a meaning as a set of types that are solutions of the constraints.

  *(Hence two constraints with the same meaning may have completely different shapes, e.g. two unsolvable constraints are equivalent)*

- Constraints are also types. As such, they can be instantiated along $\sqsubseteq$.

**Definition:**

- A presolution of a constraint $\chi$ is an instance of $\chi$ in which all constraint edge are solved.

- A solution of $\chi$ is of a the projection of a presolution of $\chi$
Semantics of constraints

Projection of a constraint

1) Remove all existential nodes, 2) garbage collect from the root node, 3) remove all dangling edges.
Projection of a constraint

1) Remove all existential nodes, 2) garbage collect from the root node, 3) remove all dangling edges.
Semantics of constraints

Projection of a constraint

1) Remove all existential nodes, 2) garbage collect from the root node, 3) remove all dangling edges.
Semantics of constraints

A unification edge $p \rightarrow q$ if $p = q$

Unification edges are solved by unification :-)

An instantiation edge $s \longrightarrow m$ is solved if the type at $m$ matches the type scheme at $s$. 
An instantiation edge $s \rightarrow m$ is solved if the type at $m$ matches the type scheme at $s$ i.e. the unification of a copy of $s$ with $m$ leaves the constraint unchanged.
An instantiation edge $s \rightarrow m$ is solved if

$$\forall p \rightarrow q \geq n \rightarrow d$$
A key property

A solvable constraint has a principal presolution, i.e. an instance of the original constraint that is a presolution and of which all other presolutions are instances.
\( \chi \Vdash \chi' \) if all solutions of \( \chi \) are solutions of \( \chi' \).
Entailment

\( \chi \vdash \chi' \) if all solutions of \( \chi \) are solutions of \( \chi' \).

In particular \( \chi \vdash \chi' \) whenever \( \chi' \sqsubseteq \chi \) (\( \chi' \) is more constrained than \( \chi \)).

However, interesting entailments are not along \( \sqsubseteq \).

\( \chi \downarrow \vdash \chi' \) if the solutions of \( \chi \) are exactly the solutions of \( \chi' \).
Sound and complete transformations

**Unif-Elim**

Solving unification edges

**Exists-Elim**

Elimination of existential nodes without inner constraints
Sound and complete transformations

**Inst-Elim-Poly** (Existential nodes and constraint edges also copied)
Sound and complete transformations

Inst-Elim-Mono  (Degenerate case

\[
\forall \vdash \forall
\]

\[
\forall \vdash \forall
\]
Sound and complete transformations

**Inst-Copy**  (Existential nodes and constraint edges also copied)
A strategy

\[ s_1 \text{ depends on } s_2 \text{ if } s_2 \rightarrow n \text{ and } n \text{ is inner } s_1. \]

A constraint \( \chi \) is admissible if the dependency relation is acyclic.
(Except for pathological cases, other cases do not have solutions).

**Strategy for solving admissible constraints**

- Independent schemes may be solved first, by **Inst-Elim-Poly**
- The unification edge that is introduced may be solved immediately.
- This way, no constraint edge is ever duplicated.
Typing constraints

\[ \exists \]
Typing example $\lambda(x)\ x$
Typing example $\lambda(x) \ x$

\[
\lambda(x) \ x \quad \Rightarrow
\]

\[
\forall \rightarrow \forall \rightarrow \bot
\]
Typing example $\lambda(x)\ x$
Typing example $\lambda(x)\ x$
Example \( \text{let } y = \lambda(x) \ x \ \text{in } y \ y \)
Example: let $y = \lambda(x)\ x$ in $y\ y$

\[
\begin{align*}
\text{let } y &= \lambda(x)\ x \\
in y\ y
\end{align*}
\]

$\Rightarrow$

\[
\begin{align*}
\text{let } y &= \lambda(x)\ x \\
in y\ y
\end{align*}
\]
Example let $y = \lambda(x) \ x$ in $y \ y$
Example let \( y = \lambda(x) \ x \) in \( y \ y \)
Example let \( y = \lambda(x) \ x \) in \( y \ y \)
Example let $y = \lambda(x) \ x$ in $y \ y$
Example let $y = \lambda(x) \; x$ in $y \; y$
Coercions: \((a : \kappa)\) is typed as \(c_{\kappa} \ a\)

\[
\forall (\alpha) \tau \quad \forall (\gamma = \forall (\alpha) \tau) \forall (\gamma' > \forall (\alpha) \tau) \gamma \rightarrow \gamma'
\]
Coercions: $(a : \kappa)$ is typed as $c_\kappa a$

\[
\exists \bar{\beta} \forall (\alpha) \tau \quad \forall (\bar{\beta}) \forall (\gamma = \forall (\alpha) \tau) \forall (\gamma' > \forall (\alpha) \tau) \gamma \rightarrow \gamma'
\]
Type soundness

We show type soundness in $\text{IML}_F$ a fully implicitly typed version of $\text{XML}_F$. 
We show type soundness in IML$^F$ a fully implicitly typed version of XML$^F$.

XML$^F$ is defined as IML$^F$, replacing $\sqsubseteq$ by $\sqsubseteq^\exists$ everywhere where $\sqsubseteq^\exists$ is $(\sqsubseteq \cup \exists)^*$

No type inference and no principal types in IML$^F$

This changes the semantics of constraints, which have more solutions. Entailment is incomparable.

Transformations rules of XML$^F$ are not complete or not sound in IML$^F$
**Type soundness**

**Subject reduction** means that $\rightarrow$ is a subrelation of $\vdash$. We show that $\vdash$ satisfies the rules defining $\rightarrow$.

**Progress** is easy.
\[ \beta \text{ preserves typings} \]

\[ (\lambda (x) \ a_1) \ a_2 \]
(λ(x) \ a_1) \ a_2 \\ \\
\equiv \\

Equivalence, by definition
\( \beta \) preserves typings

Entailment, \( \vdash \)
\( \beta \) preserves typings

Equivalence, Unification
\( \beta \) preserves typings

\[ \forall \ a_2 \Perp x \ a_1 \Perp \]

Equivalence, by existential elimination
\( \beta \) preserves typings

\[ \forall a \; \perp \; x \Rightarrow \forall a \; \perp \; x \]

Entailment, by inverse instance
\( \beta \) preserves typings

Entailment, by \textbf{Inst-Bot}, \dagger
Equivalence, decomposition of $\sqsubseteq^\beta$
Entailment, just dropping constraints
Equivalence, by definition
\( \beta \) preserves typings

Equivalence, by definition
\( \beta \) preserves typings

\[
\begin{array}{c}
\vdash \\
\not\vdash
\end{array}
\]

Enactment, by Inst-Copy, \( \dagger \)
\( \beta \) preserves typings

\[
\forall \perp. \ldots \forall \perp. \ldots
\]

Equivalence, zooming on details.
Entailment, by Inst-Bot
\[ a_1[a_2/x] \iff \begin{array}{c}
\begin{array}{c}
a_1 \\cdots \\ a_2
\end{array}
\end{array}
\]

Equivalence, by definition,
Conclusions

- Simpler, canonical definition of $\text{ML}^F$.
- Efficient, scalable type inference.
- Generalizing type constraint for ML.
  Makes type inference independent of the underlying language.
- Good basis for further extensions:
  higher-order types, recursive types, existential types, ...
- Also to be explored: semi-unification problem for $\text{ML}^F$ types.

See [http://gallium.inria.fr/~remy/mlf/](http://gallium.inria.fr/~remy/mlf/)
Appendices
Nodes/contexts are *partitioned* into four categories:

- **I**  *irreversible*
- **R^e**  *explicitly reversible*
- **R^i**  *implicitly reversible*
- **U**  *unsafe.*

They are uniquely determined by the binding tree.

- **R^i**-nodes are non-bottom nodes whose incoming binding edges all originate from other **R^i**-nodes.

- The remaining nodes are further classified by looking at the sequence of labels obtained from following their binding edges in the inverse direction (starting from the root) in the automaton drawn here.
Permissions

Relations

<table>
<thead>
<tr>
<th></th>
<th>Grafting</th>
<th>Merging</th>
<th>Raising</th>
<th>Weakening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>⊑</td>
<td>I</td>
<td>I, R</td>
<td>I</td>
</tr>
<tr>
<td>Abstraction</td>
<td>≼</td>
<td>−</td>
<td>R</td>
<td>−</td>
</tr>
<tr>
<td>Similarity</td>
<td>≼</td>
<td>−</td>
<td>R_i</td>
<td>−</td>
</tr>
</tbody>
</table>

Decompositions

We may always treat types up to ≼, since (⊑ ∪ ≼)* = (⊑; ≼)

We may also (sometimes) treat types up to ≼, since (⊑ ∪ ≻)* = (⊑; ≻)
Applying \( \forall \rightarrow \) to \( \forall \rightarrow \) returns equivalent to \( \forall \rightarrow \).

Similarly, \( \forall \rightarrow \) equivalent to \( \forall \rightarrow \).

See Gen typing rule.
A revisited syntactic presentation of $\text{ML}^F$, with an interpretation of types as sets of System-F types.

- It justifies the choice of types and type instance.
- We exhibit a correspondance with implicitly typed and explicitly typed versions of $\text{ML}^F$.
- We encodes $\text{ML}^F$ into $\text{Flet}$ (an extension of F with intersection types).

The instance relation is (slightly) enhanced by correcting artifacts of the syntactic definition in the original relation.

However, this presentation is restricted to $\text{Plain ML}^F$ (types are stratified).
A family of languages

Graphical $\text{ML}^F$

Standard $\forall \alpha$. Flexible $\forall (\alpha > \sigma)$
A family of languages

Graphical $\text{ML}^F$

$\text{F}$

$\text{ML}$

Simple Types

Standard $\forall \alpha$

Flexible $\forall (\alpha > \sigma)$
### Syntactic instance

#### Type Equivalence

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eq-Refl</td>
<td>$(Q) \sigma_1 \equiv \sigma_2$</td>
</tr>
<tr>
<td>Eq-Trans</td>
<td>$(Q) \sigma_2 \equiv \sigma_3$</td>
</tr>
<tr>
<td>Eq-Context-R</td>
<td>$(Q, \alpha \diamond \sigma) \sigma_1 \equiv \sigma_2$</td>
</tr>
<tr>
<td>Eq-Context-L</td>
<td>$(Q) \forall (\alpha \diamond \sigma) \sigma_1 \equiv \forall (\alpha \diamond \sigma) \sigma_2$</td>
</tr>
</tbody>
</table>

#### Type Abstraction

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Equiv</td>
<td>$(Q) \sigma_1 \equiv \sigma_2$</td>
</tr>
<tr>
<td>A-Trans</td>
<td>$(Q) \sigma_2 \equiv \sigma_3$</td>
</tr>
<tr>
<td>A-Context-R</td>
<td>$(Q, \alpha \diamond \sigma) \sigma_1 \equiv \sigma_2$</td>
</tr>
<tr>
<td>A-Context-L</td>
<td>$(Q) \forall (\alpha \diamond \sigma) \sigma_1 \equiv \forall (\alpha \diamond \sigma) \sigma_2$</td>
</tr>
</tbody>
</table>

#### Type Instance

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Abstract</td>
<td>$(Q) \sigma_1 \leq \sigma_2$</td>
</tr>
<tr>
<td>I-Trans</td>
<td>$(Q) \sigma_2 \leq \sigma_3$</td>
</tr>
<tr>
<td>I-Context-R</td>
<td>$(Q, \alpha \diamond \sigma) \sigma_1 \leq \sigma_2$</td>
</tr>
<tr>
<td>I-Hyp</td>
<td>$(Q) \sigma_1 \leq \alpha_1$</td>
</tr>
<tr>
<td>I-Context-L</td>
<td>$(Q) \forall (\alpha \diamond \sigma) \sigma_1 \leq \forall (\alpha \diamond \sigma) \sigma_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Bot</td>
<td>$(Q) \bot \leq \sigma$</td>
</tr>
<tr>
<td>I-Rigid</td>
<td>$(Q) \forall (\alpha &gt; \sigma_1) \sigma \leq \forall (\alpha &gt; \sigma_2) \sigma$</td>
</tr>
</tbody>
</table>
Syntactic instance

Type Equivalence

**Eq-Ref**

\[(Q) \sigma \equiv \sigma\]

**Eq-Trans**

\[
\begin{array}{c}
(Q) \sigma_1 \equiv \sigma_2 \\
(Q) \sigma_2 \equiv \sigma_3 \\
(Q) \sigma_1 \equiv \sigma_3
\end{array}
\]

**Eq-Context-R**

\[
\begin{array}{c}
(Q, \alpha \diamond \sigma) \sigma_1 \equiv \sigma_2 \\
(Q) \forall (\alpha \diamond \sigma) \sigma_1 \equiv \forall (\alpha \diamond \sigma) \sigma_2
\end{array}
\]

**Eq-Context-L**

\[
\begin{array}{c}
(Q) \sigma_1 \equiv \sigma_2 \\
(Q) \forall (\alpha \diamond \sigma_1) \sigma \equiv \forall (\alpha \diamond \sigma_2) \sigma
\end{array}
\]

**Eq-Free**

\[
\begin{array}{c}
\alpha \notin \text{ftv}(\sigma_1) \\
(Q) \forall (\alpha \diamond \sigma) \sigma_1 \equiv \sigma_1
\end{array}
\]

**Eq-Comm**

\[
\begin{array}{c}
\alpha_1 \notin \text{ftv}(\sigma_2) \\
\alpha_2 \notin \text{ftv}(\sigma_1)
\end{array}
\]

\[
\begin{array}{c}
(Q) \forall (\alpha_1 \diamond_1 \sigma_1) \forall (\alpha_2 \diamond_2 \sigma_2) \sigma \\
\equiv \forall (\alpha_2 \diamond_2 \sigma_2) \forall (\alpha_1 \diamond_1 \sigma_1) \sigma
\end{array}
\]

**Eq-Mono**

\[
\begin{array}{c}
(\alpha \diamond \sigma_0) \in Q \\
(Q) \sigma_0 \equiv \tau_0
\end{array}
\]

**Eq-Var**

\[
(Q) \forall (\alpha \diamond \sigma) \alpha \equiv \sigma
\]
# Type Abstraction

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Equiv</td>
<td>$(Q) \sigma_1 \equiv \sigma_2$</td>
</tr>
<tr>
<td></td>
<td>$(Q) \sigma_1 \equiv \sigma_2$</td>
</tr>
<tr>
<td>A-Trans</td>
<td>$(Q) \sigma_1 \sqsubseteq \sigma_2$</td>
</tr>
<tr>
<td></td>
<td>$(Q) \sigma_2 \sqsubseteq \sigma_3$</td>
</tr>
<tr>
<td>A-Context-R</td>
<td>$(Q, \alpha \diamond \sigma) \sigma_1 \sqsubseteq \sigma_2$</td>
</tr>
<tr>
<td></td>
<td>$(Q) \forall (\alpha \diamond \sigma) \sigma_1 \sqsubseteq \forall (\alpha \diamond \sigma) \sigma_2$</td>
</tr>
<tr>
<td>A-Hyp</td>
<td>$(\alpha_1 = \sigma_1) \in Q$</td>
</tr>
<tr>
<td></td>
<td>$(Q) \sigma_1 \sqsubseteq \alpha_1$</td>
</tr>
</tbody>
</table>

**A-Context-L**

$(Q) \sigma_1 \sqsubseteq \sigma_2$

$(Q) \forall (\alpha = \sigma_1) \sigma \sqsubseteq \forall (\alpha = \sigma_2) \sigma$
# Type Instance

<table>
<thead>
<tr>
<th>I-Abstract</th>
<th>I-Trans</th>
<th>I-Context-R</th>
<th>I-Hyp</th>
</tr>
</thead>
<tbody>
<tr>
<td>((Q)\sigma_1 \sqsubseteq \sigma_2)</td>
<td>((Q)\sigma_1 \leq \sigma_2)</td>
<td>((Q, \alpha \diamond \sigma)\sigma_1 \leq \sigma_2)</td>
<td>((\alpha_1 \geq \sigma_1) \in Q)</td>
</tr>
<tr>
<td>((Q)\sigma_1 \leq \sigma_2)</td>
<td>((Q)\sigma_1 \leq \sigma_3)</td>
<td>((Q)\forall(\alpha \diamond \sigma)\sigma_1 \leq \forall(\alpha \diamond \sigma)\sigma_2)</td>
<td>((Q)\sigma_1 \leq \alpha_1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I-Context-L</th>
<th>I-Bot</th>
<th>I-Rigid</th>
</tr>
</thead>
<tbody>
<tr>
<td>((Q)\sigma_1 \leq \sigma_2)</td>
<td>((Q)\bot \leq \sigma)</td>
<td>((Q)\forall(\alpha &gt; \sigma_1)\sigma \leq \forall(\alpha = \sigma_1)\sigma)</td>
</tr>
<tr>
<td>((Q)\forall(\alpha &gt; \sigma_1)\sigma \leq \forall(\alpha &gt; \sigma_2)\sigma)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Typing $\lambda(y : \forall(\alpha) \alpha \to \alpha) \ y \ y$
Typing \( \lambda(y : \forall(\alpha) \alpha \rightarrow \alpha) \; y \; y \)

\[
\begin{align*}
\lambda(y : \forall(\alpha) \alpha \rightarrow \alpha) \; y \; y & \quad \Rightarrow \quad \lambda(y) \text{ let } y = c_{\kappa \text{id}} \; y \; \text{ in } y \; y \\
\end{align*}
\]

(1) by definition
Typing $\lambda(y : \forall(\alpha) \alpha \to \alpha) \; y \; y$

(2) by definition

$\lambda(y) \; \text{let } y = c_{\kappa_{\text{id}}} \; y \; \text{in } y \; y$
Typing $\lambda(y : \forall(\alpha) \alpha \rightarrow \alpha) \ y \ y$

(3) unification
Typing $\lambda(y : \forall(\alpha) \alpha \rightarrow \alpha) y y$

(4) existential elimination
Typing \( \lambda(y : \forall(\alpha) \alpha \rightarrow \alpha) \ y \ y \)
Typing $\lambda(y : \forall(\alpha) \alpha \rightarrow \alpha) \ y \ y$

(6) by definition
(7) many steps
The End