Qualified Types for ML-F

Daan Leijen and Andres Löh
27 September 2005

Motivation / contribution

Motivation:

- Make ML-F suitable for use in a full-fledged programming language (read: Haskell).

Contribution:

- Extend ML-F with support for qualified types.
- Give an evidence translation of qualified ML-F types into a core language.

Overview

1. Hindley-Milner and ML-F
   - Arbitrary-rank polymorphism
   - Impredicativity
2. Qualified types
   - Type classes
3. ML-F with qualified types
   - Example/Problem
   - Solution

Overview

1. Hindley-Milner and ML-F
   - Arbitrary-rank polymorphism
   - Impredicativity
2. Qualified types
   - Type classes
3. ML-F with qualified types
   - Example/Problem
   - Solution

Hindley-Milner

- The type system we all know and love.
- At the basis of ML, Haskell, Clean, and many other functional programming languages.
- Efficient type inference.
- No type annotations required.
- Principal types.

ML-F

- ML-F is an extension of the Hindley-Milner type system (ICFP 2003).
- Arbitrary-rank polymorphism.
- Impredicative.
- Type annotations are required where higher-rank polymorphic values are introduced.
- (Still) Principal types.

Arbitrary-rank polymorphism
≈ functions can have polymorphic arguments

$\mathsf{f\ choose} = (\mathsf{choose\ True\ False}, \mathsf{choose}\ \texttt{'a'}\ \texttt{'b'})$

- Within f, the function choose is used at two different types.
  The first occurrence is of type $\mathsf{Bool} \rightarrow \mathsf{Bool} \rightarrow \alpha$.
  The second occurrence is of type $\mathsf{Char} \rightarrow \mathsf{Char} \rightarrow \alpha$.
- The above definition does not type-check in Haskell (nor in ML).

In ML-F:

$\mathsf{f}\ (\mathsf{choose} :: \forall \alpha.\, \alpha \rightarrow \alpha \rightarrow \alpha) = (\mathsf{choose\ True\ False}, \mathsf{choose}\ \texttt{'a'}\ \texttt{'b'})$

Arbitrary-rank polymorphism
$\approx$ functions can have polymorphic arguments

> f choose = (choose True False, choose 'a' 'b')

- Within f, the function choose is used at two different types.
  The first occurrence is of type $\text{Bool} \rightarrow \text{Bool} \rightarrow \alpha$.
  The second occurrence is of type $\text{Char} \rightarrow \text{Char} \rightarrow \alpha$.
- The above definition does not type-check in Haskell (nor in ML).

In ML-F:

> f (choose :: $\forall\alpha.\ \alpha \rightarrow \alpha \rightarrow \alpha$) = (choose True False, choose 'a' 'b')

Arbitrary-rank polymorphism
≈ functions can have polymorphic arguments

$\mathsf{f}\ \mathsf{choose} = (\mathsf{choose}\ \mathsf{True}\ \mathsf{False}, \mathsf{choose}\ \texttt{'a'}\ \texttt{'b'})$

- Within f, the function choose is used at two different types.
  The first occurrence is of type $\mathsf{Bool} \rightarrow \mathsf{Bool} \rightarrow \alpha$.
  The second occurrence is of type $\mathsf{Char} \rightarrow \mathsf{Char} \rightarrow \alpha$.
- The above definition does not type-check in Haskell (nor in ML).

In ML-F:

$\mathsf{f}\ (\mathsf{choose} :: \forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha) = (\mathsf{choose}\ \mathsf{True}\ \mathsf{False}, \mathsf{choose}\ \texttt{'a'}\ \texttt{'b'})$

Arbitrary-rank polymorphism
≈ functions can have polymorphic arguments

$$\mathsf{f}\ \mathsf{choose} = (\mathsf{choose}\ \mathsf{True}\ \mathsf{False}, \mathsf{choose}\ \texttt{'a'}\ \texttt{'b'})$$

- Within f, the function choose is used at two different types.
  The first occurrence is of type $\mathsf{Bool} \rightarrow \mathsf{Bool} \rightarrow \alpha$.
  The second occurrence is of type $\mathsf{Char} \rightarrow \mathsf{Char} \rightarrow \alpha$.
- The above definition does not type-check in Haskell (nor in ML).

In ML-F:

$$\mathsf{f}\ (\mathsf{choose} :: \forall\alpha.\, \alpha \rightarrow \alpha \rightarrow \alpha) = (\mathsf{choose}\ \mathsf{True}\ \mathsf{False}, \mathsf{choose}\ \texttt{'a'}\ \texttt{'b'})$$

Arbitrary-rank polymorphism ≈ functions can have polymorphic arguments

$\text{f choose} = (\text{choose True False}, \text{choose } \texttt{'a' 'b'})$

- Within f, the function choose is used at two different types. The first occurrence is of type $\text{Bool} \to \text{Bool} \to \alpha$. The second occurrence is of type $\text{Char} \to \text{Char} \to \alpha$.
- The above definition does not type-check in Haskell (nor in ML).

In ML-F:

$\text{f (choose} :: \forall\alpha.\, \alpha \to \alpha \to \alpha) = (\text{choose True False}, \text{choose } \texttt{'a' 'b'})$

ML-F type:

$(\forall\alpha.\, \alpha \to \alpha \to \alpha) \to (\text{Bool}, \text{Char})$

Arbitrary-rank polymorphism
≈ functions can have polymorphic arguments

$$\text{f choose} = (\text{choose True False}, \text{choose } \texttt{'a'}\ \texttt{'b'})$$

- Within f, the function choose is used at two different types. The first occurrence is of type $\text{Bool} \to \text{Bool} \to \alpha$. The second occurrence is of type $\text{Char} \to \text{Char} \to \alpha$.
- The above definition does not type-check in Haskell (nor in ML).

In ML-F:

$$\text{f (choose} :: \forall\alpha.\, \alpha \to \alpha \to \alpha) = (\text{choose True False}, \text{choose } \texttt{'a'}\ \texttt{'b'})$$

Real ML-F type:

$$\forall(\alpha = \forall\beta.\, \beta \to \beta \to \beta).\, \alpha \to (\text{Bool}, \text{Char})$$

Impredicativity
$\approx$ quantified variables range over polymorphic types

choose :: $\forall\alpha.\, \alpha \to \alpha \to \alpha$
id :: $\forall\beta.\, \beta \to \beta$

choose id :: ...

Possibility 2 (id is used at its polymorphic type):

choose id :: $(\forall\beta.\, \beta \to \beta) \to (\forall\beta.\, \beta \to \beta)$

ML-F:

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

Impredicativity
$\approx$ quantified variables range over polymorphic types

choose :: $\forall \alpha. \, \alpha \to \alpha \to \alpha$
id :: $\forall \beta. \, \beta \to \beta$

Possibility 1 (predicative, Haskell):

choose id :: $\forall \gamma. \, (\gamma \to \gamma) \to (\gamma \to \gamma)$

Possibility 2 (id is used at its polymorphic type):

choose id :: $(\forall \beta. \, \beta \to \beta) \to (\forall \beta. \, \beta \to \beta)$

ML-F:

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

Impredicativity
$\approx$ quantified variables range over polymorphic types

choose :: $\forall\alpha.\, \alpha \to \alpha \to \alpha$
id :: $\forall\beta.\, \beta \to \beta$

Possibility 1 (predicative, Haskell):

choose id :: $\forall\gamma.\, (\gamma \to \gamma) \to (\gamma \to \gamma)$

Possibility 2 (id is used at its polymorphic type):

choose id :: $(\forall\beta.\, \beta \to \beta) \to (\forall\beta.\, \beta \to \beta)$

ML-F:

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

Impredicativity
$\approx$ quantified variables range over polymorphic types

$$\begin{aligned} \text{choose} &:: \forall\alpha.\, \alpha \to \alpha \to \alpha \\ \text{id} &:: \forall\beta.\, \beta \to \beta \end{aligned}$$

Possibility 1 (predicative, Haskell):

$$\text{choose id} :: \forall\gamma.\, (\gamma \to \gamma) \to (\gamma \to \gamma)$$

Possibility 2 (id is used at its polymorphic type):

$$\text{choose id} :: (\forall\beta.\, \beta \to \beta) \to (\forall\beta.\, \beta \to \beta)$$

ML-F:

$$\text{choose id} :: \forall(\alpha \geq \forall\beta.\, \beta \to \beta).\, \alpha \to \alpha$$

Impredicativity
$\approx$ parametrized datatypes can be instantiated to polymorphic types

[id] :: $\forall(\alpha \geq \forall\beta.\,\beta \rightarrow \beta).\,[\alpha]$

Alternatively:

[id] :: $[\forall\beta.\,\beta \rightarrow \beta]$

Haskell:

[id] :: $\forall\beta.\,[\beta \rightarrow \beta]$

First-class higher-rank polymorphism

f :: $\forall(\alpha = \forall\beta.\, \beta \rightarrow \beta \rightarrow \beta).\, \alpha \rightarrow (\mathsf{Bool}, \mathsf{Char})$

[f]

id f

[runST]

runST computation vs. runST $ computation

First-class higher-rank polymorphism

f :: $\forall(\alpha = \forall\beta.\, \beta \rightarrow \beta \rightarrow \beta).\, \alpha \rightarrow (\mathsf{Bool}, \mathsf{Char})$

[f]

id f

[runST]

runST computation vs. runST $ computation

The ML-F type language

Monotypes:

$$\tau ::= \mathsf{g}\ \tau_1 \ldots \tau_n \mid \alpha$$

Polytypes:

$$\sigma ::= \bot \mid \forall Q.\, \tau$$

Prefix:

$$Q ::= \varepsilon \mid (\alpha \diamond \sigma)\ Q$$

Bounds:

$$\diamond ::= \geqslant \mid =$$

The notation $\forall \alpha.\, \tau$ abbreviates $\forall (\alpha \geq \bot).\, \tau$.

Overview

1. Hindley-Milner and ML-F
   - Arbitrary-rank polymorphism
   - Impredicativity
2. Qualified types
   - Type classes
3. ML-F with qualified types
   - Example/Problem
   - Solution

Qualified types

- A general framework for types with predicates π.
- Usually written $\pi \Rightarrow \sigma$.
- Many applications:
  - Type classes
  - Implicit parameters
  - Records (has-predicates, lacks-predicates)
- Generic theory by Mark Jones, and many others (rules for predicate entailment and propagation).
- Implementation: usually using evidence translation.

Type classes

- Predicates of the form $\mathsf{C}\ \tau$.
- Example:

  $(==) :: \forall\alpha.\, \mathsf{Eq}\ \alpha \Rightarrow \alpha \rightarrow \alpha \rightarrow \mathsf{Bool}$

- Predicates assert that certain types are instances of a class.
- Evidence: a dictionary of the class methods for the type in question.

Evidence translation

- Predicates are represented by evidence.
- Evidence for class predicates is a dictionary containing the class methods.
- Evidence is automatically provided or propagated.

original code	internal translation
$(==) :: \forall\alpha.\, \mathsf{Eq}\ \alpha \Rightarrow \alpha \rightarrow \alpha \rightarrow \mathsf{Bool}$	$(==) :: \forall\alpha.\, \mathsf{Eq}\ \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \mathsf{Bool}$
'a' == 'b'	$(==)\ \mathsf{eq}_{\mathsf{Char}}$ 'a' 'b'
$\mathsf{elem} :: \forall\alpha.\, \mathsf{Eq}\ \alpha \Rightarrow [\alpha] \rightarrow \mathsf{Bool}$ $\mathsf{elem}\ y = \mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, x == y)$	$\mathsf{elem} :: \forall\alpha.\, \mathsf{Eq}\ \alpha \rightarrow [\alpha] \rightarrow \mathsf{Bool}$ $\mathsf{elem}\ \mathsf{eq}_{\alpha}\ y =$ $\quad \mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, (==)\ \mathsf{eq}_{\alpha}\ x\ y)$

Evidence translation

- Predicates are represented by evidence.
- Evidence for class predicates is a dictionary containing the class methods.
- Evidence is automatically provided or propagated.

original code	internal translation
$(==) :: \forall\alpha.\, \mathsf{Eq}\ \alpha \Rightarrow \alpha \to \alpha \to \mathsf{Bool}$	$(==) :: \forall\alpha.\, \mathsf{Eq}\ \alpha \to \alpha \to \alpha \to \mathsf{Bool}$
'a' == 'b'	$(==)\ \mathsf{eq}_{\mathsf{Char}}$ 'a' 'b'
$\mathsf{elem} :: \forall\alpha.\, \mathsf{Eq}\ \alpha \Rightarrow [\alpha] \to \mathsf{Bool}$ $\mathsf{elem}\ y = \mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, x == y)$	$\mathsf{elem} :: \forall\alpha.\, \mathsf{Eq}\ \alpha \to [\alpha] \to \mathsf{Bool}$ $\mathsf{elem}\ \mathsf{eq}_\alpha\ y =$ $\quad \mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, (==)\ \mathsf{eq}_\alpha\ x\ y)$

Evidence translation

- Predicates are represented by evidence.
- Evidence for class predicates is a dictionary containing the class methods.
- Evidence is automatically provided or propagated.

original code	internal translation
$(==) :: \forall\alpha.\, \mathsf{Eq}\ \alpha \Rightarrow \alpha \rightarrow \alpha \rightarrow \mathsf{Bool}$	$(==) :: \forall\alpha.\, \mathsf{Eq}\ \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \mathsf{Bool}$
$\text{'a'} == \text{'b'}$	$(==)\ \mathsf{eq}_{\mathsf{Char}}\ \text{'a'}\ \text{'b'}$
$\mathsf{elem} :: \forall\alpha.\, \mathsf{Eq}\ \alpha \Rightarrow [\alpha] \rightarrow \mathsf{Bool}$ $\mathsf{elem}\ y = \mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, x == y)$	$\mathsf{elem} :: \forall\alpha.\, \mathsf{Eq}\ \alpha \rightarrow [\alpha] \rightarrow \mathsf{Bool}$ $\mathsf{elem}\ \mathsf{eq}_\alpha\ y =$ $\quad \mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, (==)\ \mathsf{eq}_\alpha\ x\ y)$

Evidence translation

- Predicates are represented by evidence.
- Evidence for class predicates is a dictionary containing the class methods.
- Evidence is automatically provided or propagated.

original code	internal translation
$(==) :: \forall\alpha.\, \mathsf{Eq}\ \alpha \Rightarrow \alpha \to \alpha \to \mathsf{Bool}$	$(==) :: \forall\alpha.\, \mathsf{Eq}\ \alpha \to \alpha \to \alpha \to \mathsf{Bool}$
`'a' == 'b'`	$(==)\ \mathsf{eq}_{\mathsf{Char}}$ `'a'` `'b'`
$\mathsf{elem} :: \forall\alpha.\, \mathsf{Eq}\ \alpha \Rightarrow [\alpha] \to \mathsf{Bool}$	$\mathsf{elem} :: \forall\alpha.\, \mathsf{Eq}\ \alpha \to [\alpha] \to \mathsf{Bool}$
$\mathsf{elem}\ y = \mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, x == y)$	$\mathsf{elem}\ \mathsf{eq}_\alpha\ y =$ $\mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, (==)\ \mathsf{eq}_\alpha\ x\ y)$

Evidence translation

- Predicates are represented by evidence.
- Evidence for class predicates is a dictionary containing the class methods.
- Evidence is automatically provided or propagated.

original code	internal translation
$(==) :: \forall\alpha.\,\mathsf{Eq}\ \alpha \Rightarrow \alpha \rightarrow \alpha \rightarrow \mathsf{Bool}$	$(==) :: \forall\alpha.\,\mathsf{Eq}\ \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \mathsf{Bool}$
`'a' == 'b'`	$(==)\ \mathsf{eq}_{\mathsf{Char}}$ `'a'` `'b'`
$\mathsf{elem} :: \forall\alpha.\,\mathsf{Eq}\ \alpha \Rightarrow [\alpha] \rightarrow \mathsf{Bool}$ $\mathsf{elem}\ y = \mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, x == y)$	$\mathsf{elem} :: \forall\alpha.\,\mathsf{Eq}\ \alpha \rightarrow [\alpha] \rightarrow \mathsf{Bool}$ $\mathsf{elem}\ \mathsf{eq}_{\alpha}\ y =$ $\quad \mathsf{or} \cdot \mathsf{map}\ (\lambda x.\, (==)\ \mathsf{eq}_{\alpha}\ x\ y)$

ML-F and qualified types

When adding qualified types to ML-F, the tricky part is to adapt the evidence translation.

Overview

1 Hindley-Milner and ML-F
- Arbitrary-rank polymorphism
- Impredicativity

2 Qualified types
- Type classes

3 ML-F with qualified types
- Example/Problem
- Solution

Example: four lists

$$\begin{aligned}
xs_1 &= [\,] \\
xs_2 &= const : xs_1 \\
xs_3 &= min \quad : xs_2 \\
xs_4 &= (<) \quad : xs_3
\end{aligned}$$

$$\begin{aligned}
const &:: \forall \alpha\, \beta.\, \alpha \to \beta \to \alpha \\
min &:: \forall \alpha.\, Ord\ \alpha \Rightarrow \alpha \to \alpha \to \alpha \\
(<) &:: \forall \alpha.\, Ord\ \alpha \Rightarrow \alpha \to \alpha \to Bool
\end{aligned}$$

Example: four lists

$$
\begin{aligned}
xs_1 &= [\,] & &:: \forall\alpha.\,[\alpha] \\
xs_2 &= \mathsf{const} : xs_1 & &:: \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha] \\
xs_3 &= \mathsf{min} \quad\; : xs_2 & &:: \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow [\alpha \to \alpha \to \alpha] \\
xs_4 &= (<) \quad\; : xs_3 & &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

Haskell types.

$$
\begin{aligned}
\mathsf{const} &:: \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha \\
\mathsf{min} &:: \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \to \alpha \to \alpha \\
(<) &:: \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \to \alpha \to \mathsf{Bool}
\end{aligned}
$$

Example: four lists

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 && :: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \rightarrow \beta \rightarrow \alpha).\,[\gamma] \\
\mathsf{xs}_3 &= \mathsf{min} \quad : \mathsf{xs}_2 && :: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \rightarrow \alpha \rightarrow \alpha).\,[\gamma] \\
\mathsf{xs}_4 &= (<) \quad : \mathsf{xs}_3 && :: [\mathsf{Bool} \rightarrow \mathsf{Bool} \rightarrow \mathsf{Bool}]
\end{aligned}
$$

ML-F types.

$$
\begin{aligned}
\mathsf{const} &:: \forall\alpha\,\beta.\,\alpha \rightarrow \beta \rightarrow \alpha \\
\mathsf{min} &:: \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \rightarrow \alpha \rightarrow \alpha \\
(<) &:: \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \rightarrow \alpha \rightarrow \mathsf{Bool}
\end{aligned}
$$

Haskell evidence translation

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] && :: \forall\alpha.\,[\alpha]\\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 && :: \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha]\\
\mathsf{xs}_3 &= \mathsf{min} \quad : \mathsf{xs}_2 && :: \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow [\alpha \to \alpha \to \alpha]\\
\mathsf{xs}_4 &= (<) \quad : \mathsf{xs}_3 && :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_1^* &:: \forall\alpha.\,[\alpha]\\
\mathsf{xs}_1^* &= [\,]^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_2^* &:: \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha]\\
\mathsf{xs}_2^* &= \mathsf{const}^* : \mathsf{xs}_1^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_3^* &:: \forall\alpha.\,\mathsf{Ord}\ \alpha \to [\alpha \to \alpha \to \alpha]\\
\mathsf{xs}_3^* &= \lambda\mathsf{ord}_\alpha.\,\mathsf{min}^*\ \mathsf{ord}_\alpha : \mathsf{xs}_2^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]\\
\mathsf{xs}_4^* &= (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \mathsf{xs}_3^*\ \mathsf{ord}_{\mathsf{Bool}}
\end{aligned}
$$

Haskell evidence translation

$$
\begin{array}{llll}
\mathsf{xs}_1 = [\,] & & :: & \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 = \mathsf{const} & : \mathsf{xs}_1 & :: & \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha] \\
\mathsf{xs}_3 = \mathsf{min} & : \mathsf{xs}_2 & :: & \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow [\alpha \to \alpha \to \alpha] \\
\mathsf{xs}_4 = (<) & : \mathsf{xs}_3 & :: & [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_1^* :: \ \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* = [\,]^*
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_2^* :: \ \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha] \\
\mathsf{xs}_2^* = \mathsf{const}^* : \mathsf{xs}_1^*
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_3^* :: \ \forall\alpha.\,\mathsf{Ord}\ \alpha \to [\alpha \to \alpha \to \alpha] \\
\mathsf{xs}_3^* = \lambda\mathsf{ord}_\alpha.\,\mathsf{min}^*\ \mathsf{ord}_\alpha : \mathsf{xs}_2^*
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_4^* :: \ [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* = (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \mathsf{xs}_3^*\ \mathsf{ord}_{\mathsf{Bool}}
\end{array}
$$

Haskell evidence translation

$$\begin{aligned}
xs_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
xs_2 &= const : xs_1 && :: \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha] \\
xs_3 &= min \quad : xs_2 && :: \forall\alpha.\,\mathrm{Ord}\ \alpha \Rightarrow [\alpha \to \alpha \to \alpha] \\
xs_4 &= (<) \quad : xs_3 && :: [\mathrm{Bool} \to \mathrm{Bool} \to \mathrm{Bool}]
\end{aligned}$$

$$\begin{aligned}
xs_1^* &:: \forall\alpha.\,[\alpha] \\
xs_1^* &= [\,]^*
\end{aligned}$$

$$\begin{aligned}
xs_2^* &:: \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha] \\
xs_2^* &= const^* : xs_1^*
\end{aligned}$$

$$\begin{aligned}
xs_3^* &:: \forall\alpha.\,\mathrm{Ord}\ \alpha \to [\alpha \to \alpha \to \alpha] \\
xs_3^* &= \lambda ord_\alpha.\, min^*\ ord_\alpha : xs_2^*
\end{aligned}$$

$$\begin{aligned}
xs_4^* &:: [\mathrm{Bool} \to \mathrm{Bool} \to \mathrm{Bool}] \\
xs_4^* &= (<)^*\ ord_{\mathrm{Bool}} : xs_3^*\ ord_{\mathrm{Bool}}
\end{aligned}$$

Haskell evidence translation

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 && :: \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha] \\
\mathsf{xs}_3 &= \mathsf{min} \quad : \mathsf{xs}_2 && :: \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow [\alpha \to \alpha \to \alpha] \\
\mathsf{xs}_4 &= (<) \quad : \mathsf{xs}_3 && :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_1^* &:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* &= [\,]^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_2^* &:: \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha] \\
\mathsf{xs}_2^* &= \mathsf{const}^* : \mathsf{xs}_1^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_3^* &:: \forall\alpha.\,\mathsf{Ord}\ \alpha \to [\alpha \to \alpha \to \alpha] \\
\mathsf{xs}_3^* &= \lambda\mathsf{ord}_\alpha.\,\mathsf{min}^*\ \mathsf{ord}_\alpha : \mathsf{xs}_2^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* &= (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \mathsf{xs}_3^*\ \mathsf{ord}_{\mathsf{Bool}}
\end{aligned}
$$

Haskell evidence translation

$$\begin{aligned}
xs_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
xs_2 &= \mathsf{const} : xs_1 && :: \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha] \\
xs_3 &= \mathsf{min} \quad : xs_2 && :: \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow [\alpha \to \alpha \to \alpha] \\
xs_4 &= (<) \quad : xs_3 && :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}$$

$$\begin{aligned}
xs_1^* &:: \forall\alpha.\,[\alpha] \\
xs_1^* &= [\,]^*
\end{aligned}$$

$$\begin{aligned}
xs_2^* &:: \forall\alpha\,\beta.\,[\alpha \to \beta \to \alpha] \\
xs_2^* &= \mathsf{const}^* : xs_1^*
\end{aligned}$$

$$\begin{aligned}
xs_3^* &:: \forall\alpha.\,\mathsf{Ord}\ \alpha \to [\alpha \to \alpha \to \alpha] \\
xs_3^* &= \lambda \mathsf{ord}_\alpha.\,\mathsf{min}^*\ \mathsf{ord}_\alpha : xs_2^*
\end{aligned}$$

$$\begin{aligned}
xs_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
xs_4^* &= (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : xs_3^*\ \mathsf{ord}_{\mathsf{Bool}}
\end{aligned}$$

Naïve evidence translation for ML-F types

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 && :: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 &= \mathsf{min} : \mathsf{xs}_2 && :: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 &= (<) : \mathsf{xs}_3 && :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{xs}_1^* :: \forall\alpha.\,[\alpha] \\
&\mathsf{xs}_1^* = [\,]^*
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{xs}_2^* :: [\forall\alpha\,\beta.\,\alpha \to \beta \to \alpha] \\
&\mathsf{xs}_2^* = \mathsf{const}^* : \mathsf{xs}_1^*
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{xs}_3^* :: [\forall\alpha.\,\mathsf{Ord}\ \alpha \to \alpha \to \alpha \to \alpha] \\
&\mathsf{xs}_3^* = \mathsf{min}^* : \ldots \mathsf{xs}_2^* \ldots
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{xs}_4^* :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
&\mathsf{xs}_4^* = (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \ldots \mathsf{xs}_3^* \ldots
\end{aligned}
$$

Naïve evidence translation for ML-F types

$$
\begin{array}{lll}
\mathsf{xs}_1 = [\,] & & :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 = \mathsf{const} & : \mathsf{xs}_1 & :: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 = \mathsf{min} & : \mathsf{xs}_2 & :: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 = (<) & : \mathsf{xs}_3 & :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_1^* :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* = [\,]^*
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_2^* :: [\forall\alpha\,\beta.\,\alpha \to \beta \to \alpha] \\
\mathsf{xs}_2^* = \mathsf{const}^* : \mathsf{xs}_1^*
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_3^* :: [\forall\alpha.\,\mathsf{Ord}\ \alpha \to \alpha \to \alpha \to \alpha] \\
\mathsf{xs}_3^* = \mathsf{min}^* : \ldots \mathsf{xs}_2^* \ldots
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_4^* :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* = (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \ldots \mathsf{xs}_3^* \ldots
\end{array}
$$

Naïve evidence translation for ML-F types

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 && :: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 &= \mathsf{min} \quad : \mathsf{xs}_2 && :: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 &= (<) \quad : \mathsf{xs}_3 && :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_1^* &:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* &= [\,]^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_2^* &:: [\forall\alpha\,\beta.\,\alpha \to \beta \to \alpha] \\
\mathsf{xs}_2^* &= \mathsf{const}^* : \mathsf{xs}_1^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_3^* &:: [\forall\alpha.\,\mathsf{Ord}\ \alpha \to \alpha \to \alpha \to \alpha] \\
\mathsf{xs}_3^* &= \mathsf{min}^* : \ldots \mathsf{xs}_2^* \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* &= (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \ldots \mathsf{xs}_3^* \ldots
\end{aligned}
$$

Naïve evidence translation for ML-F types

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 && :: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 &= \mathsf{min} \quad : \mathsf{xs}_2 && :: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 &= (<) \quad : \mathsf{xs}_3 && :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_1^* &:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* &= [\,]^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_2^* &:: [\forall\alpha\,\beta.\,\alpha \to \beta \to \alpha] \\
\mathsf{xs}_2^* &= \mathsf{const}^* : \mathsf{xs}_1^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_3^* &:: [\forall\alpha.\,\mathsf{Ord}\ \alpha \to \alpha \to \alpha \to \alpha] \\
\mathsf{xs}_3^* &= \mathsf{min}^* : \ldots \mathsf{xs}_2^* \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* &= (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \ldots \mathsf{xs}_3^* \ldots
\end{aligned}
$$

Naïve evidence translation for ML-F types

$$
\begin{array}{ll}
\mathsf{xs}_1 = [\,] & :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 = \mathsf{const} : \mathsf{xs}_1 & :: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 = \mathsf{min} \quad : \mathsf{xs}_2 & :: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\;\alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 = (<) \quad : \mathsf{xs}_3 & :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_1^* :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* = [\,]^*
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_2^* :: [\forall\alpha\,\beta.\,\alpha \to \beta \to \alpha] \\
\mathsf{xs}_2^* = \mathsf{const}^* : \mathsf{xs}_1^*
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_3^* :: [\forall\alpha.\,\mathsf{Ord}\;\alpha \to \alpha \to \alpha \to \alpha] \\
\mathsf{xs}_3^* = \mathsf{min}^* : \ldots \mathsf{xs}_2^* \ldots
\end{array}
$$

$$
\begin{array}{l}
\mathsf{xs}_4^* :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* = (<)^*\;\mathsf{ord}_{\mathsf{Bool}} : \ldots \mathsf{xs}_3^* \ldots
\end{array}
$$

Idea

$f :: \forall(\alpha \geq \sigma).\, \tau$

$f = \ldots x_\sigma \ldots x_\sigma \ldots$

View

$\forall(\alpha \geq \sigma).\, \tau$ as $\forall\alpha.\, \alpha \geq \sigma \Rightarrow \tau$

$f^* :: \forall\alpha.\, (\sigma^* \to \alpha) \to \tau^*$

$f^* = \lambda v.\, \ldots (v\ x_\sigma) \ldots (v\ x_\sigma) \ldots$

Idea

$f :: \forall(\alpha \geq \sigma).\, \tau$

$f = \ \ldots x_\sigma \ldots x_\sigma \ldots$

View

$\forall(\alpha \geq \sigma).\, \tau$ as $\forall\alpha.\, \alpha \geq \sigma \Rightarrow \tau$

$f^* :: \forall\alpha.\, (\sigma^* \to \alpha) \to \tau^*$

$f^* = \lambda v.\ \ldots (v\ x_\sigma) \ldots (v\ x_\sigma) \ldots$

Idea

$\mathsf{f} :: \forall(\alpha \geq \sigma).\, \tau$

$\mathsf{f} = \ \ldots \mathsf{x}_\sigma \ldots \mathsf{x}_\sigma \ldots$

View

$\forall(\alpha \geq \sigma).\, \tau$ as $\forall\alpha.\, \alpha \geq \sigma \Rightarrow \tau$

$\mathsf{f}^* :: \forall\alpha.\, (\sigma^* \to \alpha) \to \tau^*$

$\mathsf{f}^* = \lambda\mathsf{v}.\ \ldots (\mathsf{v}\ \mathsf{x}_\sigma) \ldots (\mathsf{v}\ \mathsf{x}_\sigma) \ldots$

Idea

$f :: \forall(\alpha \geq \sigma).\, \tau$

$f = \ \ldots x_\sigma \ldots x_\sigma \ldots$

View

$\forall(\alpha \geq \sigma).\, \tau$ as $\forall\alpha.\, \alpha \geq \sigma \Rightarrow \tau$

$f^* :: \forall\alpha.\, (\sigma^* \to \alpha) \to \tau^*$

$f^* = \lambda v.\ \ldots (v\ x_\sigma) \ldots (v\ x_\sigma) \ldots$

Idea

$$\mathsf{f} :: \forall(\alpha \geq \sigma).\, \tau$$

$$\mathsf{f} = \ \ldots \mathsf{x}_\sigma \ldots \mathsf{x}_\sigma \ldots$$

View

$$\forall(\alpha \geq \sigma).\, \tau \quad \text{as} \quad \forall\alpha.\, \alpha \geq \sigma \Rightarrow \tau$$

$$\mathsf{f}^* :: \forall\alpha.\, (\sigma^* \to \alpha) \to \tau^*$$

$$\mathsf{f}^* = \lambda \mathsf{v}.\ \ldots (\mathsf{v}\ \mathsf{x}_\sigma) \ldots (\mathsf{v}\ \mathsf{x}_\sigma) \ldots$$

Applying the idea to the example

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] & &:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 & &:: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 &= \mathsf{min} : \mathsf{xs}_2 & &:: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 &= (<) : \mathsf{xs}_3 & &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_1^* &:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* &= [\,]^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_2^* &:: \forall\gamma.\,(\forall\alpha\,\beta.\,(\alpha \to \beta \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_2^* &= \lambda \mathsf{v}_2.\,(\mathsf{v}_2\ \mathsf{const}^*) : \mathsf{xs}_1^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_3^* &:: \forall\gamma.\,(\forall\alpha.\,(\mathsf{Ord}\ \alpha \to \alpha \to \alpha \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_3^* &= \lambda \mathsf{v}_3.\,(\mathsf{v}_3\ \mathsf{min}^*) : \mathsf{xs}_2^*\ (\lambda \mathsf{x}.\,\mathsf{v}_3\ (\lambda \mathsf{ord}_\alpha.\,\mathsf{x}))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* &= (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \mathsf{xs}_3^*\ (\lambda \mathsf{x}.\,\mathsf{x}\ \mathsf{ord}_{\mathsf{Bool}})
\end{aligned}
$$

Applying the idea to the example

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 && :: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 &= \mathsf{min} : \mathsf{xs}_2 && :: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 &= (<) : \mathsf{xs}_3 && :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_1^* &:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* &= [\,]^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_2^* &:: \forall\gamma.\,(\forall\alpha\,\beta.\,(\alpha \to \beta \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_2^* &= \lambda \mathsf{v}_2.\,(\mathsf{v}_2\ \mathsf{const}^*) : \mathsf{xs}_1^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_3^* &:: \forall\gamma.\,(\forall\alpha.\,(\mathsf{Ord}\ \alpha \to \alpha \to \alpha \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_3^* &= \lambda \mathsf{v}_3.\,(\mathsf{v}_3\ \mathsf{min}^*) : \mathsf{xs}_2^*\ (\lambda \mathsf{x}.\,\mathsf{v}_3\ (\lambda \mathsf{ord}_\alpha.\,\mathsf{x}))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* &= (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \mathsf{xs}_3^*\ (\lambda \mathsf{x}.\,\mathsf{x}\ \mathsf{ord}_{\mathsf{Bool}})
\end{aligned}
$$

Applying the idea to the example

$$\begin{aligned}
\mathsf{xs}_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 && :: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 &= \mathsf{min} \quad : \mathsf{xs}_2 && :: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\;\alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 &= (<) \quad : \mathsf{xs}_3 && :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}$$

$$\begin{aligned}
\mathsf{xs}_1^* &:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* &= [\,]^*
\end{aligned}$$

$$\begin{aligned}
\mathsf{xs}_2^* &:: \forall\gamma.\,(\forall\alpha\,\beta.\,(\alpha \to \beta \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_2^* &= \lambda \mathsf{v}_2.\,(\mathsf{v}_2\;\mathsf{const}^*) : \mathsf{xs}_1^*
\end{aligned}$$

$$\begin{aligned}
\mathsf{xs}_3^* &:: \forall\gamma.\,(\forall\alpha.\,(\mathsf{Ord}\;\alpha \to \alpha \to \alpha \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_3^* &= \lambda \mathsf{v}_3.\,(\mathsf{v}_3\;\mathsf{min}^*) : \mathsf{xs}_2^*\;(\lambda \mathsf{x}.\,\mathsf{v}_3\;(\lambda \mathsf{ord}_\alpha.\,\mathsf{x}))
\end{aligned}$$

$$\begin{aligned}
\mathsf{xs}_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* &= (<)^*\;\mathsf{ord}_{\mathsf{Bool}} : \mathsf{xs}_3^*\;(\lambda \mathsf{x}.\,\mathsf{x}\;\mathsf{ord}_{\mathsf{Bool}})
\end{aligned}$$

Applying the idea to the example

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] &&:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 &&:: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 &= \mathsf{min} : \mathsf{xs}_2 &&:: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\ \alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 &= (<) : \mathsf{xs}_3 &&:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_1^* &:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* &= [\,]^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_2^* &:: \forall\gamma.\,(\forall\alpha\,\beta.\,(\alpha \to \beta \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_2^* &= \lambda \mathsf{v}_2.\,(\mathsf{v}_2\ \mathsf{const}^*) : \mathsf{xs}_1^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_3^* &:: \forall\gamma.\,(\forall\alpha.\,(\mathsf{Ord}\ \alpha \to \alpha \to \alpha \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_3^* &= \lambda \mathsf{v}_3.\,(\mathsf{v}_3\ \mathsf{min}^*) : \mathsf{xs}_2^*\ (\lambda \mathsf{x}.\,\mathsf{v}_3\ (\lambda \mathsf{ord}_\alpha.\,\mathsf{x}))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* &= (<)^*\ \mathsf{ord}_{\mathsf{Bool}} : \mathsf{xs}_3^*\ (\lambda \mathsf{x}.\,\mathsf{x}\ \mathsf{ord}_{\mathsf{Bool}})
\end{aligned}
$$

Applying the idea to the example

$$
\begin{aligned}
\mathsf{xs}_1 &= [\,] && :: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_2 &= \mathsf{const} : \mathsf{xs}_1 && :: \forall(\gamma \geq \forall\alpha\,\beta.\,\alpha \to \beta \to \alpha).\,[\gamma] \\
\mathsf{xs}_3 &= \mathsf{min} \quad : \mathsf{xs}_2 && :: \forall(\gamma \geq \forall\alpha.\,\mathsf{Ord}\;\alpha \Rightarrow \alpha \to \alpha \to \alpha).\,[\gamma] \\
\mathsf{xs}_4 &= (<) \quad : \mathsf{xs}_3 && :: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}]
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_1^* &:: \forall\alpha.\,[\alpha] \\
\mathsf{xs}_1^* &= [\,]^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_2^* &:: \forall\gamma.\,(\forall\alpha\,\beta.\,(\alpha \to \beta \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_2^* &= \lambda \mathsf{v}_2.\,(\mathsf{v}_2\;\mathsf{const}^*) : \mathsf{xs}_1^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_3^* &:: \forall\gamma.\,(\forall\alpha.\,(\mathsf{Ord}\;\alpha \to \alpha \to \alpha \to \alpha) \to \gamma) \to [\gamma] \\
\mathsf{xs}_3^* &= \lambda \mathsf{v}_3.\,(\mathsf{v}_3\;\mathsf{min}^*) : \mathsf{xs}_2^*\;(\lambda \mathsf{x}.\,\mathsf{v}_3\;(\lambda \mathsf{ord}_\alpha.\,\mathsf{x}))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{xs}_4^* &:: [\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}] \\
\mathsf{xs}_4^* &= (<)^*\;\mathsf{ord}_{\mathsf{Bool}} : \mathsf{xs}_3^*\;(\lambda \mathsf{x}.\,\mathsf{x}\;\mathsf{ord}_{\mathsf{Bool}})
\end{aligned}
$$

Discussion

- We can perform an evidence translation for qualified ML-F types.
- The paper contains many additional details of the extension.
- ML-F with qualified types has advantages over current Haskell extensions:
  - Impredicativity makes polymorphic values truly first-class.
  - Polymorphic datastructures without explicit packing and unpacking.
  - Predicates can have polymorphic arguments, too (example: implicit parameters of polymorphic type).
- ML-F could be a type system for Haskell.
- We are working on a prototype implementation in the Morrow compiler.