Structural Subtyping as Parametric Polymorphism

Wenhao Tang [1] Daniel Hillerström [2] James McKinna [3] Michel Steuwer [4,1] Ornela Dardha [5]
Rongxiao Fu [1] Sam Lindley [1]

OOPSLA'23, Cascais, Portugal, 26th Oct 2023

1 THE UNIVERSITY of EDINBURGH
2 HUAWEI
3 HERIOT WATT UNIVERSITY
4 Technische Universität Berlin
5 University of Glasgow

Structural Subtyping vs Parametric Polymorphism

Record/Variant Subtyping vs Row/Presence Polymorphism

Structural Subtyping vs Parametric Polymorphism

Record / Variant Subtyping vs Row / Presence Polymorphism

"The definitional power of these two systems is incomparable." [1]

[1] Mitchell Wand, "Complete type infernece for simple objects", 1987

Structural Subtyping vs Parametric Polymorphism

Record/Variant Subtyping vs Row/Presence Polymorphism

"The definitional power of these two systems is incomparable." [1]

"Row variables and subtyping complement one another." [2]

[1] Mitchell Wand, "Complete type infernece for simple objects", 1987
[2] François Pottier, "Type inference in the presence of subtyping", 1998

Structural Subtyping vs Parametric Polymorphism

Record/Variant Subtyping vs Row/Presence Polymorphism

"The definitional power of these two systems is incomparable." [1]

"Row variables and subtyping complement one another." [2]

"Subtyping is in fact a poor man's row polymorphism." [3]

......

[1] Mitchell Wand, "Complete type infernece for simple objects", 1987
[2] François Pottier, "Type inference in the presence of subtyping", 1998
[3] Andreas Rossberg, "lambda-the-ultimate.org/node/3711#comment-52984", 2009

Structural Subtyping vs Parametric Polymorphism

Record/Variant Subtyping ➡️ Row/Presence Polymorphism

"The definitional power of these two systems is incomparable." [1]

"Row variables and subtyping complement one another." [2]

"Subtyping is in fact a poor man's row polymorphism." [3]

......

[1] Mitchell Wand, "Complete type infernece for simple objects", 1987
[2] François Pottier, "Type inference in the presence of subtyping", 1998
[3] Andreas Rossberg, "lambda-the-ultimate.org/node/3711#comment-52984", 2009

Let's start with *simple* variant subtyping!

λ

Let's start with *simple* variant subtyping!

$\lambda_{[\,]}$

λ

$$
\begin{aligned}
\mathit{age} &: [\mathsf{Age} : \mathtt{Int}] \\
\mathit{age} &= (\mathsf{Age}\ 42)^{[\mathsf{Age}:\mathtt{Int}]} \\
\mathit{getAge} &: [\mathsf{Age} : \mathtt{Int}; \mathsf{Year} : \mathtt{Int}] \longrightarrow \mathtt{Int} \\
\mathit{getAge} &= \lambda x^{[\mathsf{Age}:\mathtt{Int};\mathsf{Year}:\mathtt{Int}]}.\ \mathbf{case}\ x\ \{\mathsf{Age}\ y \mapsto y; \mathsf{Year}\ y \mapsto 2023 - y\}
\end{aligned}
$$

Let's start with *simple* variant subtyping!

$\lambda_{[\,]}$ ······· $\lambda^{\leqslant}_{[\,]}$

λ

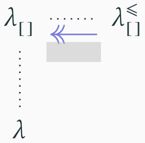
$$\begin{aligned}
\mathit{age} &: [\mathsf{Age} : \mathsf{Int}] \\
\mathit{age} &= (\mathsf{Age}\ 42)^{[\mathsf{Age}:\mathsf{Int}]} \\
\mathit{getAge} &: [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}] \to \mathsf{Int} \\
\mathit{getAge} &= \lambda x^{[\mathsf{Age}:\mathsf{Int};\mathsf{Year}:\mathsf{Int}]}.\ \mathbf{case}\ x\ \{\mathsf{Age}\ y \mapsto y; \mathsf{Year}\ y \mapsto 2023 - y\}
\end{aligned}$$

$$\mathit{getAge}\ (\mathit{age} \triangleright [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}]) \in \lambda^{\leqslant}_{[\,]}$$

Variant subtyping allows *extension* of labels.

$\mathit{age} \triangleright [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}]$	simple subtyping
$(\lambda x^{\mathsf{Unit}}.\ \mathit{age}) \triangleright (\mathsf{Unit} \to [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}])$	strictly covariant subtyping
$\mathit{getAge} \triangleright ([\mathsf{Age} : \mathsf{Int}] \to \mathsf{Int})$	full subtyping

Let's start with *simple* variant subtyping!



$$
\begin{aligned}
\mathit{age} &: [\mathsf{Age} : \mathsf{Int}] \\
\mathit{age} &= (\mathsf{Age}\ 42)^{[\mathsf{Age}:\mathsf{Int}]} \\
\mathit{getAge} &: [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}] \to \mathsf{Int} \\
\mathit{getAge} &= \lambda x^{[\mathsf{Age}:\mathsf{Int};\mathsf{Year}:\mathsf{Int}]}.\ \mathbf{case}\ x\ \{\mathsf{Age}\ y \mapsto y; \mathsf{Year}\ y \mapsto 2023 - y\}
\end{aligned}
$$

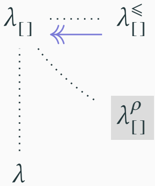
$$\mathit{getAge}\ (\mathit{age} \rhd [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}]) \in \lambda^{\leqslant}_{[]}$$

local term-involved

$$\mathit{getAge}\ (\mathbf{case}\ \mathit{age}\ \{\mathsf{Age}\ y \mapsto (\mathsf{Age}\ y)^{[\mathsf{Age}:\mathsf{Int};\mathsf{Year}:\mathsf{Int}]}\}) \in \lambda_{[]}$$

Encoding extension by destruction and reconstruction.

Let's start with *simple* variant subtyping!



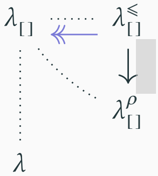
$$\begin{aligned}
&\mathit{age} &&: \quad [\mathsf{Age} : \mathsf{Int}] \\
&\mathit{age} &&= \quad (\mathsf{Age}\ 42)^{[\mathsf{Age}:\mathsf{Int}]} \\
&\mathit{getAge} &&: \quad [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}] \longrightarrow \mathsf{Int} \\
&\mathit{getAge} &&= \quad \lambda x^{[\mathsf{Age}:\mathsf{Int};\mathsf{Year}:\mathsf{Int}]}.\ \mathbf{case}\ x\ \{\mathsf{Age}\ y \mapsto y; \mathsf{Year}\ y \mapsto 2023 - y\}
\end{aligned}$$

$$\mathit{getAge}\ (\mathit{age} \rhd [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}]) \in \lambda^{\leqslant}_{[\,]}$$

$$\mathit{age}' = \Lambda\rho.\ (\mathsf{Age}\ 42)^{[\mathsf{Age}:\mathsf{Int};\,\rho\,]}$$

Row polymorphism allows *extension* of labels.

Let's start with *simple* variant subtyping!



$$
\begin{aligned}
\textit{age} &: [\mathsf{Age} : \mathsf{Int}] \\
\textit{age} &= (\mathsf{Age}\ 42)^{[\mathsf{Age}:\mathsf{Int}]} \\
\textit{getAge} &: [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}] \longrightarrow \mathsf{Int} \\
\textit{getAge} &= \lambda x^{[\mathsf{Age}:\mathsf{Int};\mathsf{Year}:\mathsf{Int}]}.\ \mathbf{case}\ x\ \{\mathsf{Age}\ y \mapsto y; \mathsf{Year}\ y \mapsto 2023 - y\}
\end{aligned}
$$

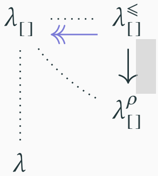
$$\textit{getAge}\ (\textit{age} \triangleright [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}]) \in \lambda^{\leqslant}_{[\,]}$$

$\downarrow$ local type-only

$$\textit{getAge}'\ (\Lambda\rho.\ \textit{age}'\ (\mathsf{Year} : \mathsf{Int}; \rho)) \in \lambda^{\rho}_{[\,]}$$

Encoding extension by type application and abstraction.

Let's start with *simple* variant subtyping!



$$\begin{array}{lcl} \mathit{age} & : & [\mathsf{Age} : \mathsf{Int}] \\ \mathit{age} & = & (\mathsf{Age}\ 42)^{[\mathsf{Age}:\mathsf{Int}]} \\ \mathit{getAge} & : & [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}] \longrightarrow \mathsf{Int} \\ \mathit{getAge} & = & \lambda x^{[\mathsf{Age}:\mathsf{Int};\mathsf{Year}:\mathsf{Int}]}.\ \mathbf{case}\ x\ \{\mathsf{Age}\ y \mapsto y; \mathsf{Year}\ y \mapsto 2023 - y\} \end{array}$$

$$\mathit{getAge}\ (\mathit{age} \triangleright [\mathsf{Age} : \mathsf{Int}; \mathsf{Year} : \mathsf{Int}]) \in \lambda^{\leqslant}_{[\,]}$$

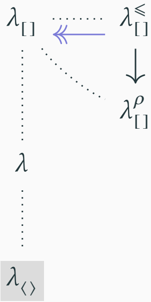
$$\Big\downarrow \text{local type-only}$$

$$\mathit{getAge}'\ (\Lambda\rho.\ \mathit{age}'\ (\mathsf{Year} : \mathsf{Int}; \rho)) \in \lambda^{\rho}_{[\,]}$$

where

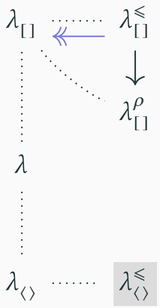
$$\begin{array}{ll} \mathit{age}' & = \Lambda\rho.\ (\mathsf{Age}\ 42)^{[\mathsf{Age}:\mathsf{Int};\rho]} \\ \mathit{getAge}' & = \lambda x^{\forall\rho.[\mathsf{Age}:\mathsf{Int};\mathsf{Year}:\mathsf{Int};\rho]}.\ \mathbf{case}\ (x\ \cdot)\ \{\ldots\} \end{array}$$ *higher-rank* polymorphism

Similarly, *simple* record subtyping.



$$
\begin{aligned}
\mathit{alice} \quad &: \quad \langle \mathsf{Name} : \mathtt{String}; \mathsf{Age} : \mathtt{Int} \rangle \\
\mathit{alice} \quad &= \quad \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle \\
\mathit{getName} \quad &: \quad \langle \mathsf{Name} : \mathtt{String} \rangle \longrightarrow \mathtt{String} \\
\mathit{getName} \quad &= \quad \lambda x^{\langle \mathsf{Name}:\mathtt{String} \rangle}.\ (x.\mathsf{Name})
\end{aligned}
$$

Similarly, *simple* record subtyping.

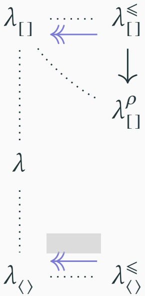


$$\begin{aligned}
\mathit{alice} &: \langle \mathsf{Name} : \mathtt{String}; \mathsf{Age} : \mathtt{Int} \rangle \\
\mathit{alice} &= \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle \\
\mathit{getName} &: \langle \mathsf{Name} : \mathtt{String} \rangle \longrightarrow \mathtt{String} \\
\mathit{getName} &= \lambda x^{\langle \mathsf{Name}:\mathtt{String} \rangle}.\ (x.\mathsf{Name})
\end{aligned}$$

$$\mathit{getName}\ (\mathit{alice} \rhd \langle \mathsf{Name} : \mathtt{String} \rangle) \in \lambda^{\leqslant}_{\langle\rangle}$$

Record subtyping allows *restriction* of labels.

Similarly, *simple* record subtyping.



$$\begin{aligned}
\textit{alice} &: \langle \mathsf{Name} : \mathtt{String}; \mathsf{Age} : \mathtt{Int} \rangle \\
\textit{alice} &= \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle \\
\textit{getName} &: \langle \mathsf{Name} : \mathtt{String} \rangle \longrightarrow \mathtt{String} \\
\textit{getName} &= \lambda x^{\langle \mathsf{Name}:\mathtt{String} \rangle}.\ (x.\mathsf{Name})
\end{aligned}$$

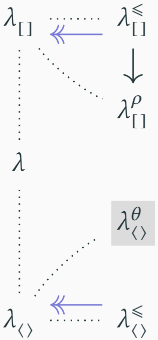
$$\textit{getName}\ (\textit{alice} \rhd \langle \mathsf{Name} : \mathtt{String} \rangle) \in \lambda^{\leqslant}_{\langle\rangle}$$

↓ local term-involved

$$\textit{getName}\ \langle \mathsf{Name} = \textit{alice}.\mathsf{Name} \rangle \in \lambda_{\langle\rangle}$$

Encoding restriction by destruction and reconstruction.

Similarly, *simple* record subtyping.



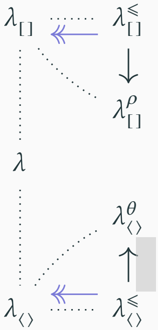
$$
\begin{aligned}
alice &: \langle \mathsf{Name} : \mathsf{String}; \mathsf{Age} : \mathsf{Int} \rangle \\
alice &= \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle \\
getName &: \langle \mathsf{Name} : \mathsf{String} \rangle \longrightarrow \mathsf{String} \\
getName &= \lambda x^{\langle \mathsf{Name} : \mathsf{String} \rangle}.\ (x.\mathsf{Name})
\end{aligned}
$$

$$getName\ (alice \triangleright \langle \mathsf{Name} : \mathsf{String} \rangle) \in \lambda^{\leqslant}_{\langle\rangle}$$

$$alice' = \Lambda \theta_1 \theta_2.\ \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle^{\langle \mathsf{Name}^{\theta_1} : \mathsf{String}; \mathsf{Age}^{\theta_2} : \mathsf{Int} \rangle}$$

Presence polymorphism allows *restriction* of labels.

Similarly, *simple* record subtyping.



$$\begin{aligned} \textit{alice} &: \langle \text{Name} : \texttt{String}; \text{Age} : \texttt{Int} \rangle \\ \textit{alice} &= \langle \text{Name} = \text{"Alice"}; \text{Age} = 42 \rangle \\ \textit{getName} &: \langle \text{Name} : \texttt{String} \rangle \longrightarrow \texttt{String} \\ \textit{getName} &= \lambda x^{\langle \text{Name}:\texttt{String} \rangle}.\ (x.\text{Name}) \end{aligned}$$

$$\textit{getName}\ (\textit{alice} \rhd \langle \text{Name} : \texttt{String} \rangle) \in \lambda^{\leqslant}_{\langle\rangle}$$

$$\Big\downarrow \text{local type-only}$$

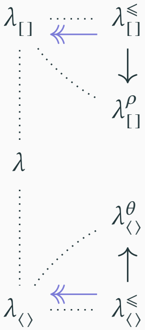
$$\textit{getName}'\ (\Lambda\theta.\ \textit{alice}'\ \theta \circ)$$

where

$$\textit{alice}' = \Lambda\theta_1\theta_2.\ \langle \ldots \rangle^{\langle \text{Name}^{\theta_1}:\texttt{String};\text{Age}^{\theta_2}:\texttt{Int} \rangle}$$

$$\textit{getName}' = \lambda x^{\forall\theta.\langle \text{Name}^{\theta}:\texttt{String} \rangle}.\ ((x\ \bullet).\text{Name})$$ *higher-rank* polymorphism

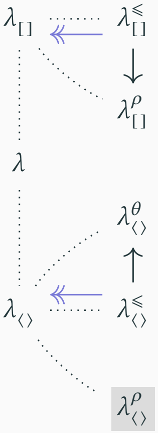
What if we swap the polymorphism?



Short summary:

row polymorphism	➡️	extension	⬅️	variant subtyping
presence polymorphism	➡️	restriction	⬅️	record subtyping

What if we swap the polymorphism?

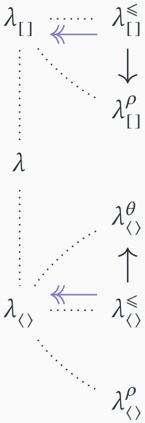


Short summary:

row polymorphism	➡️	extension	⬅️	variant subtyping
presence polymorphism	➡️	restriction	⬅️	record subtyping

Can we use row polymorphism for records?

What if we swap the polymorphism?



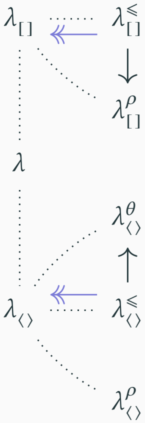
Short summary:

row polymorphism	➡	extension	⬅	variant subtyping
presence polymorphism	➡	restriction	⬅	record subtyping

Can we use row polymorphism for records?

restriction on covariant positions ↔ *extension on contravariant positions.*

What if we swap the polymorphism?



Short summary:

row polymorphism	➡	extension	⬅	variant subtyping
presence polymorphism	➡	restriction	⬅	record subtyping

Can we use row polymorphism for records?

restriction on covariant positions ↔ *extension on contravariant positions.*

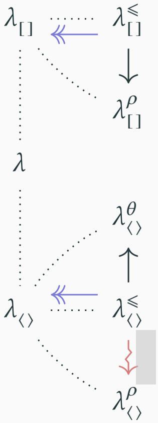
$$\lambda^{\leqslant}_{\langle\rangle} \xrightarrow{???} \lambda^{\rho}_{\langle\rangle}$$

$$[\![alice]\!] = \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle$$

$$[\![getName]\!] = \Lambda\rho.\ \lambda x^{\langle \mathsf{Name}:\mathtt{String};\ \rho\ \rangle}.\ (x.\mathsf{Name})$$

$$\left[\!\!\left[\begin{array}{l} getName \\ \quad (alice \rhd \langle \mathsf{Name} : \mathtt{String} \rangle) \end{array} \right]\!\!\right] = [\![getName]\!]\ (\mathsf{Age} : \mathtt{Int})\ [\![alice]\!]$$

What if we swap the polymorphism?



Short summary:

row polymorphism	➡	extension	⬅	variant subtyping
presence polymorphism	➡	restriction	⬅	record subtyping

Can we use row polymorphism for records?

restriction on covariant positions ↔ *extension on contravariant positions.*

$$\lambda^{\leqslant}_{\langle\rangle} \;\rightsquigarrow\!\!\!\!\times\; \lambda^{\rho}_{\langle\rangle}$$

$$[\![alice]\!] = \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle$$

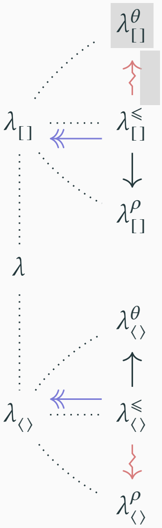
$$[\![getName]\!] = \Lambda\rho.\, \lambda x^{\langle \mathsf{Name:String};\, \rho \rangle}.\, (x.\mathsf{Name})$$

$$[\![getName\ (alice \triangleright \langle \mathsf{Name} : \mathsf{String} \rangle)]\!] = [\![getName]\!]\ \text{~~(Age : Int)~~}\ [\![alice]\!]$$

non-compositional

No type-only translation exists.

What if we swap the polymorphism?



Short summary:

row polymorphism	➡	extension	⬅	variant subtyping
presence polymorphism	➡	restriction	⬅	record subtyping

Can we use row polymorphism for records?

restriction on covariant positions ↔ *extension on contravariant positions.*

$\lambda^{\leqslant}_{\langle\rangle}$ ❎ $\rightsquigarrow$ $\lambda^{\rho}_{\langle\rangle}$

$[\![alice]\!] = \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle$

$[\![getName]\!] = \Lambda\rho\,.\,\lambda x^{\langle \mathtt{Name:String};\, \rho \rangle}.\,(x.\mathsf{Name})$

$[\![getName\ (alice \rhd \langle \mathsf{Name} : \mathtt{String} \rangle)]\!] = [\![getName]\!]\ \sout{(\mathtt{Age : Int})}\ [\![alice]\!]$

non-compositional

No type-only translation exists. Similar for variants.

Let type inference guess.

Non-compositional due to the lack of information $\texttt{Age} : \texttt{Int}$.

$$
\begin{aligned}
[\![\mathit{alice}]\!] &= \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle \\
[\![\mathit{getName}]\!] &= \Lambda\rho.\ \lambda x^{\langle \mathsf{Name}:\texttt{String};\rho\rangle}.\ (x.\mathsf{Name}) \\
\left[\!\!\left[\begin{array}{l} \mathit{getName} \\ \quad (\mathit{alice} \rhd \langle \mathsf{Name} : \texttt{String}\rangle) \end{array} \right]\!\!\right] &= [\![\mathit{getName}]\!]\ (\mathsf{Age} : \texttt{Int})\ [\![\mathit{alice}]\!]
\end{aligned}
$$

Let type inference guess.

Non-compositional due to the lack of information $\texttt{Age} : \texttt{Int}$.

$$
\begin{aligned}
[\![\mathit{alice}]\!] &= \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle \\
[\![\mathit{getName}]\!] &= \Lambda\rho.\ \lambda x^{\langle \mathsf{Name}:\texttt{String};\rho \rangle}.\ (x.\mathsf{Name}) \\
\left[\!\!\left[\begin{array}{l} \mathit{getName} \\ \quad (\mathit{alice} \rhd \langle \mathsf{Name} : \texttt{String} \rangle) \end{array} \right]\!\!\right] &= [\![\mathit{getName}]\!]\ (\mathsf{Age} : \texttt{Int})\ [\![\mathit{alice}]\!]
\end{aligned}
$$

But type inference knows it!

$$
\begin{aligned}
[\![\mathit{alice}]\!] &= \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle &&: \langle \mathsf{Name} : \texttt{String}; \mathsf{Age} : \texttt{Int} \rangle \\
[\![\mathit{getName}]\!] &= \lambda x.\ (x.\mathsf{Name}) &&: \forall\rho.\ \langle \mathsf{Name} : \texttt{String}; \rho \rangle \longrightarrow \texttt{String} \\
\left[\!\!\left[\begin{array}{l} \mathit{getName} \\ \quad (\mathit{alice} \rhd \langle \mathsf{Name} : \texttt{String} \rangle) \end{array} \right]\!\!\right] &= [\![\mathit{getName}]\!]\ [\![\mathit{alice}]\!] &&: \texttt{String}
\end{aligned}
$$

Let type inference guess.

But type inference knows it!

$$
\begin{aligned}
[\![\mathit{alice}]\!] &= \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle &&: \langle \mathsf{Name} : \mathtt{String}; \mathsf{Age} : \mathtt{Int} \rangle \\
[\![\mathit{getName}]\!] &= \lambda x.\ (x.\mathsf{Name}) &&: \forall \rho.\ \langle \mathsf{Name} : \mathtt{String}; \rho \rangle \longrightarrow \mathtt{String} \\
\left[\!\!\left[\begin{array}{l} \mathit{getName} \\ \quad (\mathit{alice} \rhd \langle \mathsf{Name} : \mathtt{String} \rangle) \end{array} \right]\!\!\right] &= [\![\mathit{getName}]\!]\ [\![\mathit{alice}]\!] &&: \mathtt{String}
\end{aligned}
$$

This applies to full subtyping (but with rank restriction).

$$\lambda^{\leqslant \text{full}}_{\langle\rangle}$$

Let type inference guess.

But type inference knows it!

$$
\begin{aligned}
[\![\mathit{alice}]\!] &= \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle &&: \langle \mathsf{Name} : \mathtt{String}; \mathsf{Age} : \mathtt{Int} \rangle \\
[\![\mathit{getName}]\!] &= \lambda x.\ (x.\mathsf{Name}) &&: \forall \rho.\ \langle \mathsf{Name} : \mathtt{String}; \rho \rangle \longrightarrow \mathtt{String} \\
\left[\!\!\left[\begin{array}{l} \mathit{getName} \\ \quad (\mathit{alice} \rhd \langle \mathsf{Name} : \mathtt{String} \rangle) \end{array} \right]\!\!\right] &= [\![\mathit{getName}]\!]\ [\![\mathit{alice}]\!] &&: \mathtt{String}
\end{aligned}
$$

This applies to full subtyping (but with rank restriction).

$$
\begin{array}{ccc}
\lambda^{\leqslant \mathrm{full}}_{\langle\rangle} & & \\
\vdots & & \\
\lambda^{\leqslant \mathrm{full}}_{\langle\rangle 2} & \longrightarrow & \lambda^{\rho 1}_{\langle\rangle}
\end{array}
$$

Let type inference guess.

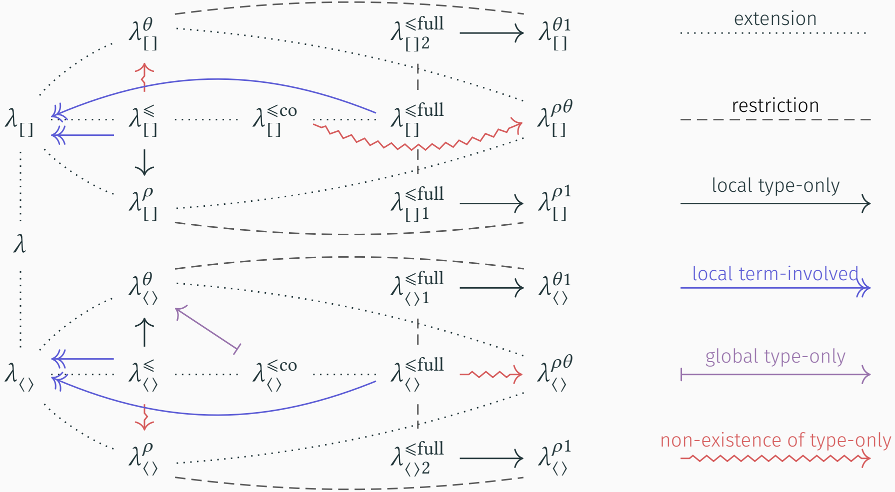
But type inference knows it!

$$
\begin{array}{lcll}
[\![\mathit{alice}]\!] & = & \langle \mathsf{Name} = \text{"Alice"}; \mathsf{Age} = 42 \rangle & : \langle \mathsf{Name} : \mathtt{String}; \mathsf{Age} : \mathtt{Int} \rangle \\
[\![\mathit{getName}]\!] & = & \lambda x.\ (x.\mathsf{Name}) & : \forall \rho.\ \langle \mathsf{Name} : \mathtt{String}; \rho \rangle \longrightarrow \mathtt{String} \\
\left[\!\!\left[\begin{array}{l} \mathit{getName} \\ \ (\mathit{alice} \rhd \langle \mathsf{Name} : \mathtt{String} \rangle) \end{array} \right]\!\!\right] & = & [\![\mathit{getName}]\!]\ [\![\mathit{alice}]\!] & : \mathtt{String}
\end{array}
$$

This applies to full subtyping (but with rank restriction).

$$
\begin{array}{ccc}
\lambda^{\leqslant \mathrm{full}}_{\langle\rangle 1} & \longrightarrow & \lambda^{\theta 1}_{\langle\rangle} \\
\vdots & & \\
\lambda^{\leqslant \mathrm{full}}_{\langle\rangle} & & \\
\vdots & & \\
\lambda^{\leqslant \mathrm{full}}_{\langle\rangle 2} & \longrightarrow & \lambda^{\rho 1}_{\langle\rangle}
\end{array}
$$

Full Picture



More in the paper: formal definitions and metatheories (type preservation & operational correspondence) of all translations and proofs of non-existence results.

Thank you!

Takeaway:

- In explicit calculi, even simple subtyping requires higher-rank polymorphism.

Thank you!

Takeaway:

- In explicit calculi, even simple subtyping requires higher-rank polymorphism.

- For simple subtyping, there is a symmetry between records and variants.
- But symmetry is broken later.

Thank you!

Takeaway:

- In explicit calculi, even simple subtyping requires higher-rank polymorphism.

- For simple subtyping, there is a symmetry between records and variants.
- But symmetry is broken later.

- No encoding of full subtyping.
- But type inference can help (OCaml).