# Pushing the Bounds of Subtyping

### **Didier** Rémy

Some ideas are based on joint works with Julien Cretin and Gabriel Scherer

# INRIA

# Luca Cardelli's Fest

September 2014

Early ideas in the 60's. First formalization by Reynolds (1980).

The Amber language (Cardelli, 1984)

Type system based on simply-typed  $\lambda\text{-calculus}$ 

Early ideas in the 60's. First formalization by Reynolds (1980). The Amber language (Cardelli, 1984)

Type system based on simply-typed  $\lambda$ -calculus + a subtyping rule:

$$\frac{\Gamma \vdash a : \tau \qquad \frac{\tau \leq \tau'}{\Gamma \vdash a : \tau'}$$

Early ideas in the 60's. First formalization by Reynolds (1980). The Amber language (Cardelli, 1984)

Type system based on simply-typed  $\lambda$ -calculus + a subtyping rule:

$$\frac{\Gamma \vdash \mathsf{a} : \tau \qquad \tau \leq \tau'}{\Gamma \vdash \mathsf{a} : \tau'}$$

Subtyping is defined by:

$$\bot \leq \tau \qquad \qquad \tau \leq \top$$

$$\frac{\tau_1' \le \tau_1 \qquad \tau_2 \le \tau_2'}{(\tau_1 \to \tau_2) \le (\tau_1' \to \tau_2')}$$

Early ideas in the 60's. First formalization by Reynolds (1980). The Amber language (Cardelli, 1984)

Type system based on simply-typed  $\lambda$ -calculus + a subtyping rule:

$$\frac{\Gamma \vdash \mathsf{a} : \tau \qquad \tau \leq \tau'}{\Gamma \vdash \mathsf{a} : \tau'}$$

Subtyping is defined by:



Early ideas in the 60's. First formalization by Reynolds (1980). The Amber language (Cardelli, 1984)

Type system based on simply-typed  $\lambda$ -calculus + a subtyping rule:

$$\frac{\Gamma \vdash \mathsf{a} : \tau \qquad \tau \leq \tau'}{\Gamma \vdash \mathsf{a} : \tau'}$$

Subtyping is defined by:

$$\bot \leq \tau \qquad \qquad \tau \leq \top$$

$$\frac{\tau_1' \le \tau_1 \quad \tau_2 \le \tau_2'}{(\tau_1 \to \tau_2) \le (\tau_1' \to \tau_2')}$$

# (For the record)

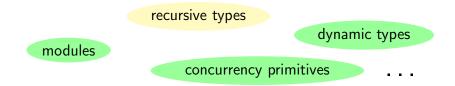
I am omitting an essential part of the story...

Amber also had records and subtyping on records that were at the basis of object encodings and inheritance.

I am omitting an essential part of the story...

Amber also had records and subtyping on records that were at the basis of object encodings and inheritance.

and many more features:



Each of which later became one of Luca's research topic

How to mix subtyping with polymorphism?

# $\forall (\alpha) \sigma$

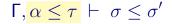
#### System F

How to mix subtyping with polymorphism: Cardelli and Wegner (1985)

$$\forall (\alpha \leq \tau) \ \sigma$$

Quest, 
$$F_{<:}$$

How to mix subtyping with polymorphism: Cardelli and Wegner (1985)



How to mix subtyping with polymorphism: Cardelli and Wegner (1985)

 $\begin{array}{cccc} \mathsf{\Gamma}, \alpha \leq \tau \ \vdash \ \sigma \leq \sigma' \\ \hline \mathsf{\Gamma} \quad \vdash \quad \forall (\alpha \leq \tau) \ \sigma & \leq \quad \forall (\alpha \leq \tau) \ \sigma' \end{array}$ 

How to mix subtyping with polymorphism: Cardelli and Wegner (1985)

$$\begin{array}{cccc} \mathsf{\Gamma}, \alpha \leq \tau \ \vdash \ \sigma \leq \sigma' \\ \hline \mathsf{\Gamma} \quad \vdash \quad \forall (\alpha \leq \tau) \ \sigma & \leq \quad \forall (\alpha \leq \tau) \ \sigma' \end{array}$$

Bounded quantification is quite expressive and at the basis of most works on type systems in the 1990's, including *A Theory of Objects*.

How to mix subtyping with polymorphism: Cardelli and Wegner (1985)

### $\label{eq:relation} \mathsf{\Gamma} \quad \vdash \quad \forall (\alpha \leq \tau) \; \sigma \quad \not\leq \quad \sigma[\alpha \leftarrow \tau']$

Still, bounded quantification is somewhat limited

- to a single, upper bound
- and does not allow instantiation of quantifiers.

How to mix subtyping with polymorphism: Cardelli and Wegner (1985)

## $\label{eq:relation} \mathsf{\Gamma} \quad \vdash \quad \forall (\alpha \leq \tau) \; \sigma \quad \not\leq \quad \sigma[\alpha \leftarrow \tau']$

Still, bounded quantification is somewhat limited

- to a single, upper bound
- and does not allow instantiation of quantifiers.

This keeps subtyping decidable\* and trackable:  $\forall \text{ is explicit, so that} \leq \text{can be left implicit.}$ 

# Type instantiation as subtyping

In ML

#### $\forall (\alpha) \ \sigma \ \leq \ \sigma[\alpha \leftarrow \tau]$

Also used in type containment (John Mitchell, 1984):

- mixes type instantiation with the Amber rules, but
- does not allow reasoning under subtyping assumptions

### Instance-bounded quantification

Introduced for partial type inference in MLF (Le Botlan&Rémy, 2003)

 $\forall (\alpha \geq \tau) \sigma$ 

Stands for the set of types  $\sigma$  where  $\alpha$  ranges over instances of  $\tau$ .

This avoids having to decide too early whether types should be instantiated or kept polymorphic.

### Can all features be combined together?

e.g.	MLF + subtyping?	$F_{<:} + Type \text{ containment } ?$
or	Restrict subtyping to equi (as in the internal languag	

# Can all features be combined together?

e.g. 
$$MLF + subtyping$$
?  $F_{<:} + Type containment$ ?

or

Restrict subtyping to equivalences? (as in the internal language of Haskell)

In fact, we also need...

- multiple bounds, e.g. as in ML with subtyping constraints.
- general, equi-recursive types and coinduction.

### Full reduction semantics

Reduction should be allowed in any context:

- for our understanding, because it should be sound to do so.
- this models reduction of open terms
- this avoids postponing type errors to type-instantiation sites
- soundness will remain true for any strategy

I also argue that

- full reduction is a better, more abstract model for the user
- CBV is a more concrete model, only needed to reason about costs

 $\tau ::= \dots | \forall (\alpha | P) \tau \qquad P ::= \tau \leq \tau | P \land P | \dots$ 

Intuitively, introduce constrained quantification

$$\tau ::= \dots | \forall (\alpha : k) \tau \qquad P ::= \tau \le \tau | P \land P | \dots \\ k ::= \star | \{\alpha | P\} | \dots$$

More conveniently, using kinds...similar to power kinds (Cardelli, 1988)

$$\tau ::= \dots | \forall (\alpha : k) \tau \qquad P ::= \tau \le \tau | P \land P | \dots \\ k ::= \star | \{\alpha | P\} | \dots$$

#### Coherence

Kinds must be inhabited. Otherwise, type errors could be hidden behind abstraction over the absurd:

$$\begin{array}{l} \label{eq:gamma} \mathsf{\Gamma}, \alpha : \{\beta : \mathit{int} \leq \mathit{bool} \rightarrow \mathit{int}\} \vdash \mathit{int} \leq \mathit{bool} \rightarrow \mathit{int} \\ \vdots \\ \hline \\ \hline \mathsf{\Gamma}, \alpha : \{\beta : \mathit{int} \leq \mathit{bool} \rightarrow \mathit{int}\} \vdash 1 \mathit{true} : \mathit{int} \\ \hline \\ \hline \mathsf{\Gamma} \vdash 1 \mathit{true} : \forall (\alpha : \{\beta \mid \mathit{int} \leq \mathit{bool} \rightarrow \mathit{int}\}) \mathit{int} \end{array}$$

Terms with incoherent constraints cannot be (safely) reduced.

$$\tau ::= \ldots \mid \forall (\alpha : k) \tau$$

$$P ::= \tau \le \tau \mid P \land P \mid \dots$$
  
$$k ::= \star \mid \{\alpha \mid P\} \mid \dots$$

#### Coherence

Kinds must be inhabited.

$$\frac{\Gamma, \alpha : \kappa \vdash \tau : \star}{\Gamma \vdash \forall (\alpha : k) \tau : \star}$$

#### Incoherence is also useful

• A kind may not be inhabited for all but only some instances of the current context—a typical situation with GADTs.

#### Incoherence is also useful

• A kind may not be inhabited for all but only some instances of the current context—a typical situation with GADTs.

#### We allow a distinct form of incoherent type abstraction

 $\forall^{\dagger}(\alpha: \mathbf{k}) \ \tau$ 

#### Incoherence is also useful

• A kind may not be inhabited for all but only some instances of the current context—a typical situation with GADTs.

We allow a distinct form of incoherent type abstraction

$$\forall^{\dagger}(\alpha:k) \tau \qquad a::=\ldots \mid \partial a$$

• Delay evaluation at the introduction of incoherent type abstraction.

$$\frac{\mathsf{\Gamma}, \alpha: \kappa \vdash \mathsf{a}: \tau \quad \mathsf{\Gamma} \vdash \sigma: \mathsf{k}}{\mathsf{\Gamma} \vdash \partial \mathsf{a}: \forall^{\dagger}(\alpha: \mathsf{k}) \tau}$$

#### Incoherence is also useful

• A kind may not be inhabited for all but only some instances of the current context—a typical situation with GADTs.

We allow a distinct form of incoherent type abstraction

$$\forall^{\dagger}(\alpha:k) \tau$$
  $a:=\ldots \mid \partial a \mid a \diamond$ 

- Delay evaluation at the introduction of incoherent type abstraction.
- Resume evaluation at its elimination.

 $\frac{\Gamma, \alpha : \kappa \vdash a : \tau \quad \Gamma \vdash \sigma : k}{\Gamma \vdash \partial a : \forall^{\dagger}(\alpha : k) \tau} \qquad \frac{\Gamma \vdash \partial a : \forall^{\dagger}(\alpha : k) \tau \quad \Gamma \vdash \sigma : k}{\Gamma, \alpha : k \vdash a \diamondsuit : \tau[\alpha \leftarrow \sigma]}$ 

# To conclude our short journey

#### We have an external language for exploring the design space

- $\oplus$  Our type system is sound,
- $\ominus$  but subject reduction does not hold.
- ⊖ Thus it is not quite suitable for
  - an internal language (by lack of subject reduction)
  - nor for a surface language (an explicitly-typed version would be very verbose)
- $\oplus$  But it does help share and separate
  - the meta-theoretical study (what can be done, safely)
  - from the practical design (what restrictions should be made)

Subtyping played an important role in Luca's early research. Luca contributed a lot to make subtyping a well-understood feature. Still, he left us a few variants and new uses of subtyping to explore.

Thank you Luca for all your inspiring works and a profusion of challenging new ideas that always kept us moving forward, faster and further.