# Subtyping via distributive lattices

Stephen Dolan

6th November, 2019

stedolan@stedolan.net

# Subtyping

# Subtyping

Subtyping gives functions better types:

$$\texttt{select}\ p\ v\ d = \texttt{if}\ (p\ v)\ \texttt{then}\ v\ \texttt{else}\ d$$

# Subtyping

Subtyping gives functions better types:

$$\texttt{select}\ p\ v\ d = \texttt{if}\ (p\ v)\ \texttt{then}\ v\ \texttt{else}\ d$$

ML says the type is:

$$\forall \alpha . (\alpha \to \mathsf{bool}) \to \alpha \to \alpha \to \alpha$$

# Data flow in `select`

$$\texttt{select}\ p\ v\ d = \texttt{if}\ (p\ v)\ \texttt{then}\ v\ \texttt{else}\ d$$

$$\begin{tikzcd}
& v \arrow[dl] \arrow[dr] & & d \arrow[dl] \\
\text{argument to } p & & \text{result} &
\end{tikzcd}$$

# More types for `select`

With subtyping, we can give a more precise type:

$$\forall \alpha, \beta \text{ where } \alpha \leq \beta.\ (\alpha \to \texttt{bool}) \to \alpha \to \beta \to \beta$$

# More types for `select`

With subtyping, we can give a more precise type:

$$\forall \alpha, \beta \text{ where } \alpha \leq \beta.\ (\alpha \to \texttt{bool}) \to \alpha \to \beta \to \beta$$

but now our subtyping definition has to include constraints.

## More types for `select`

With subtyping, we can give a more precise type:

$$\forall \alpha, \beta \text{ where } \alpha \leq \beta.\, (\alpha \to \texttt{bool}) \to \alpha \to \beta \to \beta$$

but now our subtyping definition has to include constraints.

When are two constrained types equal? When is one more general?

# What are type variables, anyway?

How do we even define subtyping with type variables?

# What are type variables, anyway?

How do we even define subtyping with type variables?

**Type variables?**

$\sigma \leq \tau$ if $\rho(\sigma) \leq \rho(\tau)$ for every $\rho$ mapping type variables to ground types.

# What are type variables, anyway?

How do we even define subtyping with type variables?

**Type variables?**

$\sigma \leq \tau$ if $\rho(\sigma) \leq \rho(\tau)$ for every $\rho$ mapping type variables to ground types.

This is a bad[1], bad[2] idea.

---

[1] *Polymorphism, Subtyping, and Type Inference in MLsub*, Dolan and Mycroft, 2017

[2] *Set-theoretic Foundation of Parametric Polymorphism and Subtyping*, Castagna and Xu, 2011

# Type variables by quantification over ground types

Is this true?[3]

$$A \to \bot \leq A \implies (\bot \to \top) \to \bot \leq A$$

[3] *Type inference in the presence of subtyping: from theory to practice*, Pottier, 1998

# Type variables by quantification over ground types

Is this true?[3]

$$A \to \bot \leq A \implies (\bot \to \top) \to \bot \leq A$$

Just applying the subtyping rules doesn't get us anywhere.

[3] *Type inference in the presence of subtyping: from theory to practice*, Pottier, 1998

# Proving Pottier's example

$$A \to \bot \leq A \implies (\bot \to \top) \to \bot \leq A$$

We prove it by case analysis on $A$.

# Proving Pottier's example

$$A \to \bot \leq A \implies (\bot \to \top) \to \bot \leq A$$

We prove it by case analysis on $A$.

If it's $\top$, then:

$$(\bot \to \top) \to \bot \leq \top = A$$

# Proving Pottier's example

$$A \to \bot \leq A \implies (\bot \to \top) \to \bot \leq A$$

We prove it by case analysis on $A$.

If it's $\top$, then:

$$(\bot \to \top) \to \bot \leq \top = A$$

Otherwise, $A \leq \bot \to \top$ and

$$\begin{aligned} & A \leq (\bot \to \top) \\ \implies & (\bot \to \top) \to \bot \leq A \to \bot \leq A \end{aligned}$$

# Proving Pottier's example

$$A \to \bot \leq A \implies (\bot \to \top) \to \bot \leq A$$

We prove it by case analysis on $A$.

If it's $\top$, then:

$$(\bot \to \top) \to \bot \leq \top = A$$

Otherwise, $A \leq \bot \to \top$ and

$$\begin{aligned} & A \leq (\bot \to \top) \\ \implies & (\bot \to \top) \to \bot \leq A \to \bot \leq A \end{aligned}$$

So it does hold, for all $A$.

# Falsifying Pottier's example

Let's add a new type of function $\tau_1 \overset{\circ}{\to} \tau_2$.

# Falsifying Pottier's example

Let's add a new type of function $\tau_1 \xrightarrow{\circ} \tau_2$.

It's a supertype of $\tau_1 \to \tau_2$

*"function that may have side effects"*

# Falsifying Pottier's example

Let's add a new type of function $\tau_1 \overset{\circ}{\to} \tau_2$.

It's a supertype of $\tau_1 \to \tau_2$

*"function that may have side effects"*

Now we have a counterexample:

$$A = (\top \overset{\circ}{\to} \bot) \overset{\circ}{\to} \bot$$

# Falsifying Pottier's example

Let's add a new type of function $\tau_1 \overset{\circ}{\to} \tau_2$.

It's a supertype of $\tau_1 \to \tau_2$

*"function that may have side effects"*

Now we have a counterexample:

$$A = (\top \overset{\circ}{\to} \bot) \overset{\circ}{\to} \bot$$

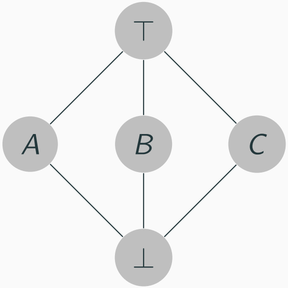
Pottier's example is true *only* by case analysis.

# Composing subtyping relations is hard

If we specify subtyping for several fragments of a language, it is difficult to compose the relations.

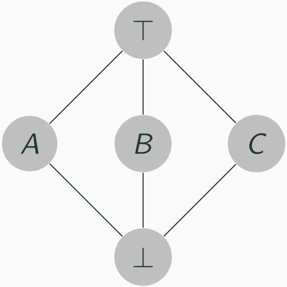
# Composing subtyping relations is hard

If we specify subtyping for several fragments of a language, it is difficult to compose the relations.



## Composing subtyping relations is hard

If we specify subtyping for several fragments of a language, it is difficult to compose the relations.




Now we have $A \leq X, B \leq X \implies C \leq X$.

Did we want this relationship between $A, B, C$?

# Deciding subtyping relations is hard

How do we go from a definition to a decision procedure?

# Deciding subtyping relations is hard

How do we go from a definition to a decision procedure?

This rule hurts a lot:

$$\frac{A \leq X \quad X \leq B}{A \leq B}$$

# Subtyping is hard...

**...to specify** (constraints, type variables)

**...to compose** (unexpected relations)

**...to decide** (non-obvious algorithms, transitivity)

# Lattices

# Lattices help to specify subtyping

A subtyping order forms a *lattice* if it has:

- **A** $\vee$ **B** The least common supertype of A and B
- **A** $\wedge$ **B** The greatest common subtype of A and B

(not necessarily union and intersection of values)

# Constraints via lattices

The lattice operators turn subtyping constraints into equations:

$$A \leq B \qquad \equiv \qquad A = A \wedge B \qquad \equiv \qquad A \vee B = B$$

# Constraints via lattices

The lattice operators turn subtyping constraints into equations:

$$A \leq B \qquad \equiv \qquad A = A \wedge B \qquad \equiv \qquad A \vee B = B$$

$$\forall \alpha, \beta \text{ where } \alpha \leq \beta.\, (\alpha \to \texttt{bool}) \to \alpha \to \beta \to \beta$$

can become

$$\forall \alpha, \beta.\, (\alpha \to \texttt{bool}) \to \alpha \to \beta \to (\alpha \vee \beta)$$

## Lattices help to compose subtyping

There is a most general way to combine two lattices, without introducing any new relations: the *lattice coproduct*.

# Lattices help to compose subtyping

There is a most general way to combine two lattices, without introducing any new relations: the *lattice coproduct*.

We can introduce a type variable $\alpha$ by taking coproducts with the three-element lattice $\{\bot, \alpha, \top\}$

# Lattices help to decide subtyping

Whitman's algorithm decides ordering in a free lattice:

$$a \wedge b \leq c \vee d \text{ iff one of: } \begin{array}{l} a \leq c \vee d \\ b \leq c \vee d \\ a \wedge b \leq c \\ a \wedge b \leq d \end{array}$$

# Lattices make subtyping easier

**...to specify** (no constraints, free lattices)

**...to compose** (coproducts)

**...to decide** (Whitman's algorithm)

# Specifying subtyping relations is still hard

How should the lattice operations interact:

- with type constructors?
  Is $A \to (B \wedge C) = (A \to B) \wedge (A \to C)$?

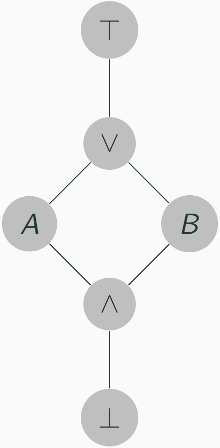
# Specifying subtyping relations is still hard

How should the lattice operations interact:

- with type constructors?
  Is $A \to (B \wedge C) = (A \to B) \wedge (A \to C)$?
- with each other?
  Is $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$?

(Some cases we get for free: $(A \vee B) \wedge A = A$ from lattice theory)

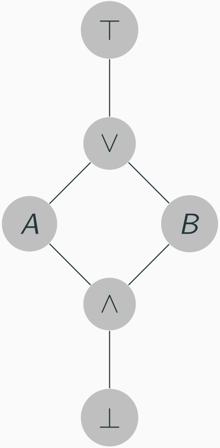
# Composing subtyping relations is still hard

,


, ?

# Composing subtyping relations is still hard


,




, ?

Lattice coproducts are weird.

# Deciding subtyping relations is still hard

$$a \wedge b \leq c \vee d \text{ if } \begin{array}{l} a \leq c \vee d \\ b \leq c \vee d \\ a \wedge b \leq c \\ a \wedge b \leq d \end{array}$$

Whitman's algorithm decides ordering in a free lattice

# Deciding subtyping relations is still hard

$$a \wedge b \leq c \vee d \text{ if } \begin{array}{l} a \leq c \vee d \\ b \leq c \vee d \\ a \wedge b \leq c \\ a \wedge b \leq d \end{array}$$

Whitman's algorithm decides ordering in a free lattice

... and only in a free lattice

# Subtyping is still hard

**...to specify** (interactions are still tricky)

**...to compose** (lattice coproduct strangeness)

**...to decide** (Whitman's does not generalise)

# Example: Intersection types

# Intersection types

BCD types[4] are of the form:

$$A ::= A \to A \mid A \wedge A \mid \top \mid \texttt{base}$$

(No type variables, no $\vee$, no $\bot$)

[4] *A Filter Lambda Model and the Completeness of Type Assignment.*, Barendregt, Coppo and Dezani-Ciancaglini, 1983

# BCD subtyping

Subtyping is a partial order with a top element:

$$\frac{}{A \leq A} \qquad \frac{A \leq B \quad B \leq C}{A \leq C} \qquad \frac{}{A \leq \top}$$

# BCD subtyping

Subtyping is a partial order with a top element:

$$\frac{}{A \leq A} \qquad \frac{A \leq B \quad B \leq C}{A \leq C} \qquad \frac{}{A \leq \top}$$

and binary meets:

$$\frac{}{A \wedge B \leq A} \qquad \frac{}{A \wedge B \leq B} \qquad \frac{}{A \leq A \wedge A} \qquad \frac{A \leq A' \quad B \leq B'}{A \wedge B \leq A' \wedge B'}$$

# BCD subtyping

Subtyping is a partial order with a top element:

$$\frac{}{A \leq A} \qquad \frac{A \leq B \quad B \leq C}{A \leq C} \qquad \frac{}{A \leq \top}$$

and binary meets:

$$\frac{}{A \wedge B \leq A} \qquad \frac{}{A \wedge B \leq B} \qquad \frac{}{A \leq A \wedge A} \qquad \frac{A \leq A' \quad B \leq B'}{A \wedge B \leq A' \wedge B'}$$

and arrow types:

$$\frac{A' \leq A \quad B \leq B'}{A \to B \leq A' \to B'} \qquad \frac{}{\top \leq \top \to \top}$$

$$\frac{}{(A \to B) \wedge (A \to C) \leq A \to (B \wedge C)}$$

## Laurent's presentation

Laurent[5] presents BCD as a relation $\Gamma \vdash A$ between a finite set of types $\Gamma$ and a type $A$.

[5] *Intersection Subtyping with Constructors*, Olivier Laurent, 2018

# Laurent's presentation

Laurent[5] presents BCD as a relation $\Gamma \vdash A$ between a finite set of types $\Gamma$ and a type $A$.

Intuitively, $\{B_1, \ldots, B_n\} \vdash A$ iff $B_1 \wedge \ldots \wedge B_n \leq A$.

"Variables" and "Weakening"

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A}$$

[5] *Intersection Subtyping with Constructors*, Olivier Laurent, 2018

# Laurent's presentation

Laurent[5] presents BCD as a relation $\Gamma \vdash A$ between a finite set of types $\Gamma$ and a type $A$.

Intuitively, $\{B_1, \ldots, B_n\} \vdash A$ iff $B_1 \wedge \ldots \wedge B_n \leq A$.

"Variables" and "Weakening"

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A}$$

Meet and $\top$:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C} \qquad \frac{}{\Gamma \vdash \top}$$

[5] *Intersection Subtyping with Constructors*, Olivier Laurent, 2018

# Laurent's presentation

Laurent[5] presents BCD as a relation $\Gamma \vdash A$ between a finite set of types $\Gamma$ and a type $A$.

Intuitively, $\{B_1, \dots, B_n\} \vdash A$ iff $B_1 \wedge \dots \wedge B_n \leq A$.

"Variables" and "Weakening"

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A}$$

Meet and $\top$:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C} \qquad \frac{}{\Gamma \vdash \top}$$

Arrow types:

$$\frac{C \vdash A_1 \ \dots \ C \vdash A_k \quad B_1, \dots, B_k \vdash D}{A_1 \to B_1, \dots, A_k \to B_k \vdash C \to D}(k \geq 1) \qquad \frac{\vdash B}{\vdash A \to B}$$

[5] *Intersection Subtyping with Constructors*, Olivier Laurent, 2018

# One theorem: Cut-elimination

**Cut is admissible**

$$\frac{\Gamma \vdash A \quad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}$$

# One theorem: Cut-elimination

**Cut is admissible**

$$\frac{\Gamma \vdash A \quad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}$$

Transitivity is a consequence.

# Subformula property

**Subformula property**

In the cut-free calculus, each type used in the conclusion is a subformula of types used in the hypothesis.

# Subformula property

**Subformula property**

In the cut-free calculus, each type used in the conclusion is a subformula of types used in the hypothesis.

**Nontriviality** $\top \nvdash \mathtt{base}$

# Subformula property

**Subformula property**

In the cut-free calculus, each type used in the conclusion is a subformula of types used in the hypothesis.

**Nontriviality** $\top \nvdash \texttt{base}$

**Inversion** If $A \to B \vdash A' \to B'$, then $A' \vdash A$, $B \vdash B'$.

# Subformula property

**Subformula property**

In the cut-free calculus, each type used in the conclusion is a subformula of types used in the hypothesis.

**Nontriviality** $\top \nvdash \texttt{base}$

**Inversion** If $A \to B \vdash A' \to B'$, then $A' \vdash A$, $B \vdash B'$.

**Decidability** Try every rule until one works or you run out.

# Decidability

The subformula property makes the decision algorithm work.

# Decidability

The subformula property makes the decision algorithm work.

Two optimisations help:

- Apply reversible rules first

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

# Decidability

The subformula property makes the decision algorithm work.

Two optimisations help:

- Apply reversible rules first

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

- Memoize repeated subderiviations

## Semi-Scott relations

Say that a relation $\Gamma \vdash A$ between a finite set of types $\Gamma$ and a type $A$ is *semi-Scott* if:

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A} \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}$$

# Semi-Scott relations

Say that a relation $\Gamma \vdash A$ between a finite set of types $\Gamma$ and a type $A$ is *semi-Scott* if:

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A} \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}$$

Every $\wedge$-semilattice gives rise to a semi-Scott relation:

$$\{B_1, \ldots, B_n\} \vdash A \text{ iff } B_1 \wedge \ldots \wedge B_n \leq A$$

# Semilattices from semi-Scott relations

Given a semi-Scott relation $\Gamma \vdash A$, define

$$\Gamma \leq \Delta \text{ iff } \forall A \in \Delta.\, \Gamma \vdash A$$

We have:

- $\Gamma \leq \Delta$ (by Var)
- $\Gamma \leq \Delta, \Delta \leq \Xi$ implies $\Gamma \leq \Xi$ (by induction on $|\Delta|$, using Weak and Cut)

Equivalence classes of $\leq$ form a semilattice.

# A recipe for Laurent-style presentations

There is a Laurent-style presentation of *every* intersection type system:

# A recipe for Laurent-style presentations

There is a Laurent-style presentation of *every* intersection type system:

The only rules to choose are those for type constructors.

The only theorem to prove is cut-elimination.

# Distributive lattices

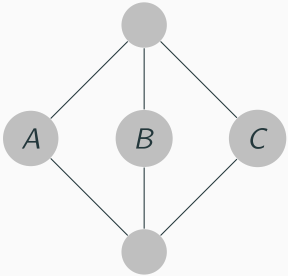
# Distributive lattices

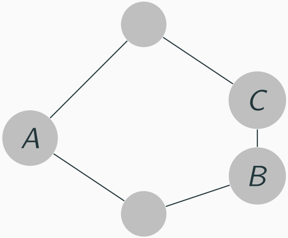
A lattice is *distributive* iff $\wedge$ and $\vee$ distribute:

$$(A \wedge B) \vee C = (A \vee C) \wedge (B \vee C)$$
$$(A \vee B) \wedge C = (A \wedge C) \vee (B \wedge C)$$

# Distributive lattices

A lattice is *distributive* iff it does not contain:



# Distributive lattices

A lattice is *distributive* iff it has an interpretation in sets:

$$\phi(A \wedge B) = \phi(A) \cap \phi(B)$$
$$\phi(A \vee B) = \phi(A) \cup \phi(B)$$

# Composing distributive lattices

The coproduct of distributive lattices is well-behaved.

Free distributive lattices exist, even complete ones.

# Scott entailment relations

Say that a relation $\Gamma \vdash \Delta$ between finite sets of types is *Scott*[6] if:

$$\frac{}{\Gamma, A \vdash A, \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma, \Gamma' \vdash \Delta, \Delta'} \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}$$

[1] *Entailment relations and distributive lattices*, Cederquist and Coquand, 2000

## Scott entailment relations

Say that a relation $\Gamma \vdash \Delta$ between finite sets of types is *Scott*[6] if:

$$\frac{}{\Gamma, A \vdash A, \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma, \Gamma' \vdash \Delta, \Delta'} \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}$$

Every distributive lattice gives rise to a Scott relation,

[1] *Entailment relations and distributive lattices*, Cederquist and Coquand, 2000

# Scott entailment relations

Say that a relation $\Gamma \vdash \Delta$ between finite sets of types is *Scott*[6] if:

$$\frac{}{\Gamma, A \vdash A, \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma, \Gamma' \vdash \Delta, \Delta'} \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}$$

Every distributive lattice gives rise to a Scott relation,
and every Scott relation to a distributive lattice.

[1] *Entailment relations and distributive lattices*, Cederquist and Coquand, 2000

# Composing distributive lattices

If $\vdash_1, \vdash_2$ present distributive lattices $D_1, D_2$,

# Composing distributive lattices

If $\vdash_1, \vdash_2$ present distributive lattices $D_1, D_2$,

then $\vdash$ presents the coproduct $D_1 + D_2$:

$$\frac{\Gamma \vdash_1 \Delta}{\mathsf{inj}_1\ \Gamma \vdash \mathsf{inj}_1\ \Delta} \qquad \frac{\Gamma \vdash_2 \Delta}{\mathsf{inj}_2\ \Gamma \vdash \mathsf{inj}_2\ \Delta}$$

# Free distributive lattice

The presentation of the free distributive lattice generated by type variables is:

# Free distributive lattice

The presentation of the free distributive lattice generated by type variables is:

$$\frac{}{\alpha \vdash \alpha}$$

# Order connectives in sequent style

The rules for $\wedge$, $\vee$ look like sequent calculus:

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta} \qquad\qquad \frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} \qquad\qquad \frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}$$

# Top and bottom in sequent style

The rules for $\bot$ and $\top$ look like False and True in sequent calculus:

$$\frac{\Gamma \vdash \Delta}{\Gamma, \top \vdash \Delta} \qquad \frac{}{\Gamma \vdash \top, \Delta}$$

$$\frac{}{\Gamma, \bot \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}$$

# Distributive lattices make subtyping easy!

**...to specify** (Scott entailment relation + cut elimination)

**...to compose** (Coproducts by combining Scott relations)

**...to decide** (Subformula property)

# Example 1: Intersection types

$$\frac{C \vdash A_1 \ \ldots \ C \vdash A_k \quad B_1, \ldots, B_k \vdash D}{A_1 \to B_1, \ldots, A_k \to B_k \vdash C \to D}(k \geq 1) \qquad \frac{\vdash B}{\vdash A \to B}$$

# Example 2: Semantic subtyping

Function subtyping[7]:

$$\frac{C \vdash A_1, \ldots, A_n \qquad \forall I' \subsetneq [1..n].\ C \vdash \{A_i \mid i \in I'\} \text{ or } \{B_i \mid i \notin I'\} \vdash D}{A_1 \to B_1, \ldots, A_n \to B_n \vdash C \to D}$$

[1] *The Relevance of Semantic Subtyping* Dezani-Ciancaglini, Frisch, Giovannetti and Motohama, 2003

# Example 2: Semantic subtyping

Function subtyping[7]:

$$\frac{C \vdash A_1, \ldots, A_n \qquad \forall I' \subsetneq [1..n].\ C \vdash \{A_i \mid i \in I'\} \text{ or } \{B_i \mid i \notin I'\} \vdash D}{A_1 \to B_1, \ldots, A_n \to B_n \vdash C \to D}$$

Negation types:

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta} \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}$$

[1] *The Relevance of Semantic Subtyping* Dezani-Ciancaglini, Frisch, Giovannetti and Motohama, 2003

# Example 3: MLsub

$$\frac{C_1, \ldots, C_m \vdash A_1, \ldots, A_n \quad B_1, \ldots, B_n \vdash D_1, \ldots, D_m}{A_1 \to B_1, \ldots, A_n \to B_n \vdash C_1 \to D_1, \ldots, C_m \to D_m}$$

# Distributive lattices make subtyping easy!

**...to specify** (Scott entailment relation + cut elimination)

**...to compose** (Coproducts by combining Scott relations)

**...to decide** (Subformula property)

# Thanks!

Questions?

`stedolan@stedolan.net`