### Polymorphism, Subtyping, and Type Inference in MLsub

Stephen Dolan and Alan Mycroft

Computer Laboratory University of Cambridge

November 8, 2016

#### The select function

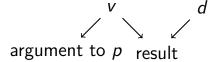
select 
$$p \ v \ d = if (p \ v)$$
 then  $v$  else  $d$ 

In ML, select has type scheme

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

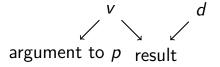
#### Data flow in select

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



#### Data flow in select

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



In MLsub, select has this type scheme:

$$\forall \alpha, \beta. (\alpha \to \mathtt{bool}) \to \alpha \to \beta \to (\alpha \sqcup \beta)$$

# Г⊢е: т

### $\Gamma \vdash \mathbf{e} : \tau$

#### Expressions of MLsub

We have functions

$$x \lambda x.e e_1 e_2$$

... and records

$$\{\ell_1=e_1,\,\ldots,\,\ell_n=e_n\}\qquad e.\ell$$

... and booleans

true false if 
$$e_1$$
 then  $e_2$  else  $e_3$ 

... and let

$$\hat{\mathbf{x}}$$
 let  $\hat{\mathbf{x}} = e_1$  in  $e_2$ 

## Γ ⊢ e : *τ*

### Typing rules of MLsub

ML + 
$$(SUB) \quad \frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash e : \tau_2} \quad \tau_1 \leq \tau_2$$

### $\Gamma \vdash e : \tau$

#### Constructing Types

The standard definition of types looks like:

$$\tau ::= \bot \mid \tau \to \tau \mid \top$$

(ignoring records and booleans for now)

#### Constructing Types

The standard definition of types looks like:

$$\tau ::= \bot \mid \tau \to \tau \mid \top$$

(ignoring records and booleans for now) with a subtyping relation like:

$$\frac{\tau_1' \le \tau_1 \quad \tau_2 \le \tau_2'}{\tau \le \tau} \quad \frac{\tau_1' \le \tau_1 \quad \tau_2 \le \tau_2'}{\tau_1 \to \tau_2 \le \tau_1' \to \tau_2'}$$

#### Lattices

#### These types form a lattice:

- ▶ least upper bounds  $\tau_1 \sqcup \tau_2$
- greatest lower bounds  $\tau_1 \sqcap \tau_2$

#### Lattices

#### These types form a lattice:

- ▶ least upper bounds  $\tau_1 \sqcup \tau_2$
- greatest lower bounds  $\tau_1 \sqcap \tau_2$

$$\frac{e_1:\tau_1\qquad e_2:\tau_2}{\text{if rand () then }e_1\text{ else }e_2:\tau_1\sqcup\tau_2}$$

Is this true, for all  $\alpha$ ?

$$\alpha \to \alpha < \bot \to \top$$

Is this true, for all  $\alpha$ ?

$$\alpha \to \alpha < \bot \to \top$$

How about this?

$$(\bot \to \top) \to \bot \le (\alpha \to \bot) \sqcup \alpha$$

Is this true, for all  $\alpha$ ?

$$\alpha \to \alpha < \bot \to \top$$

How about this?

$$(\bot \to \top) \to \bot \le (\alpha \to \bot) \sqcup \alpha$$

Yes, it turns out, by case analysis on  $\alpha$ .

Is this true, for all  $\alpha$ ?

$$\alpha \to \alpha < \bot \to \top$$

How about this?

$$(\bot \to \top) \to \bot \le (\alpha \to \bot) \sqcup \alpha$$

Yes, it turns out, by case analysis on  $\alpha$ .

And *only* by case analysis.

#### Extensibility

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

#### Extensibility

Let's add a new type of function  $\tau_1 \stackrel{\circ}{\to} \tau_2$ . It's a supertype of  $\tau_1 \to \tau_2$  "function that may have side effects"

#### Extensibility

Let's add a new type of function  $\tau_1 \stackrel{\circ}{\to} \tau_2$ . It's a supertype of  $\tau_1 \to \tau_2$  "function that may have side effects" Now we have a counterexample:

$$\alpha = (\top \xrightarrow{\circ} \bot) \xrightarrow{\circ} \bot$$

#### Extensible type systems

Two techniques give us an extensible system:

 Add explicit type variables as indeterminates gets rid of case analysis

#### Extensible type systems

Two techniques give us an extensible system:

- Add explicit type variables as indeterminates gets rid of case analysis
- Require a distributive lattice gets rid of vacuous reasoning

#### Combining types

How to combine different types into a single system?

$$\tau ::= \mathsf{bool} \mid \tau_1 \to \tau_2 \mid \{\ell_1 : \tau_1, \ldots, \ell_n : \tau_n\}$$

#### Combining types

How to combine different types into a single system?

$$\tau ::= \mathsf{bool} \mid \tau_1 \to \tau_2 \mid \{\ell_1 : \tau_1, \ldots, \ell_n : \tau_n\}$$

We should read '|' as coproduct

#### Concrete syntax

Build an actual syntax for types, by writing down all the operations on types:

$$\tau ::= \mathsf{bool} \mid \tau_1 \to \tau_2 \mid \{\ell_1 : \tau_1, \dots, \ell_n : \tau_n\} \mid \alpha \mid \top \mid \bot \mid \tau \sqcup \tau \mid \tau \sqcap \tau$$

#### Concrete syntax

Build an actual syntax for types, by writing down all the operations on types:

$$\tau ::= \mathsf{bool} \mid \tau_1 \to \tau_2 \mid \{\ell_1 : \tau_1, \dots, \ell_n : \tau_n\} \mid \alpha \mid \top \mid \bot \mid \tau \sqcup \tau \mid \tau \sqcap \tau$$

then quotient by the equations of distributive lattices, and the subtyping order.

#### Resulting types

We end up with all the standard types

#### Resulting types

We end up with all the standard types ... with the same subtyping order

#### Resulting types

We end up with all the standard types
... with the same subtyping order
... but we identify fewer of the weird types

 $\{foo: bool\} \sqcap (\top \rightarrow \top) \not\leq bool$ 

# $\Gamma \vdash \mathsf{e} : \tau$

#### Principality in ML

Intuitively, For any e typeable under  $\Gamma$ , there's a best type  $\tau$ 

#### Principality in ML

Intuitively, For any e typeable under  $\Gamma$ , there's a best type  $\tau$ 

but it's a bit more complicated than that: For any e typeable under  $\Gamma$ , there's a  $\tau$  and a substitution  $\sigma$  such that every possible typing of e under  $\Gamma$  is a substitution instance of  $\sigma\Gamma$ ,  $\tau$ .

#### Reformulating the typing rules

The complexity arises because  $\Gamma$  is part question, part answer.

#### Reformulating the typing rules

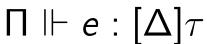
The complexity arises because  $\Gamma$  is part question, part answer.

Instead, split Γ:

- $\Delta$  maps  $\lambda$ -bound x to a type  $\tau$
- ▶  $\Pi$  maps let-bound  $\hat{\mathbf{x}}$  to a *typing schemes*  $[\Delta]\tau$







# 

## Subsumption

Define  $\leq^{\forall}$  as the least relation closed under:

- Instatiation, replacing type variables with types
- Subtyping, replacing types with supertypes

## Principality in MLsub

A principal typing scheme for e under  $\Pi$  is a  $[\Delta]\tau$  that subsumes any other.

choose takes two values and returns one of them:

choose : 
$$\forall \alpha.\alpha^1 \rightarrow \alpha^2 \rightarrow \alpha^3$$

choose takes two values and returns one of them:

choose : 
$$\forall \alpha.\alpha^1 \rightarrow \alpha^2 \rightarrow \alpha^3$$

In ML,  $\alpha^1 = \alpha^2 = \alpha^3$ . With subtyping,  $\alpha^1 \le \alpha^3$ ,  $\alpha^2 \le \alpha^3$ , but  $\alpha^1$  and  $\alpha^2$  may be incomparable.

choose takes two values and returns one of them:

choose : 
$$\forall \alpha.\alpha^1 \rightarrow \alpha^2 \rightarrow \alpha^3$$

In ML,  $\alpha^1 = \alpha^2 = \alpha^3$ . With subtyping,  $\alpha^1 \le \alpha^3$ ,  $\alpha^2 \le \alpha^3$ , but  $\alpha^1$  and  $\alpha^2$  may be incomparable.

choose : 
$$\forall \alpha \beta. \alpha \rightarrow \beta \rightarrow \alpha \sqcup \beta$$

choose takes two values and returns one of them:

choose : 
$$\forall \alpha.\alpha^1 \rightarrow \alpha^2 \rightarrow \alpha^3$$

In ML,  $\alpha^1 = \alpha^2 = \alpha^3$ . With subtyping,  $\alpha^1 \le \alpha^3$ ,  $\alpha^2 \le \alpha^3$ , but  $\alpha^1$  and  $\alpha^2$  may be incomparable.

choose : 
$$\forall \alpha \beta. \alpha \rightarrow \beta \rightarrow \alpha \sqcup \beta$$

These are equivalent ( $\equiv^{\forall}$ ): subsume each other

#### Input and output types

 $\tau \sqcup \tau'$ : produces a value which is a  $\tau$  or a  $\tau'$   $\tau \sqcap \tau'$ : requires a value which is a  $\tau$  and a  $\tau'$ 

 $\sqcup$  is for outputs, and  $\sqcap$  is for inputs.

### Input and output types

 $\tau \sqcup \tau'$ : produces a value which is a  $\tau$  or a  $\tau'$   $\tau \sqcap \tau'$ : requires a value which is a  $\tau$  and a  $\tau'$ 

 $\sqcup$  is for outputs, and  $\sqcap$  is for inputs.

Divide types into

- output types  $\tau^+$
- input types  $\tau^-$

#### Polar types

$$\tau^{+} ::= \operatorname{bool} \mid \tau_{1}^{-} \to \tau_{2}^{+} \mid \{\ell_{1} : \tau_{1}^{+}, \dots, \ell_{n} : \tau_{n}^{+}\} \mid \alpha \mid \tau_{1}^{+} \sqcup \tau_{2}^{+} \mid \bot \mid \mu \alpha. \tau^{+}$$

$$\tau^{-} ::= \operatorname{bool} \mid \tau_{1}^{+} \to \tau_{2}^{-} \mid \{\ell_{1} : \tau_{1}^{-}, \dots, \ell_{n} : \tau_{n}^{-}\} \mid \alpha \mid \tau_{1}^{-} \sqcap \tau_{2}^{-} \mid \top \mid \mu \alpha. \tau^{-}$$

#### Cases of unification

In HM inference, unification happens in three situations:

- Unifying two input types
- Unifying two output types
- Using the output of one expression as input to another

#### Cases of unification

In HM inference, unification happens in three situations:

- ► Unifying two input types Introduce ⊔
- ► Unifying two output types
  Introduce □
- Using the output of one expression as input to another

$$au^+ < au^-$$
 constraint

Suppose we have an identity function

$$\alpha \to \alpha$$

Suppose we have an identity function, which uses its argument as a  $\boldsymbol{\tau}$ 

$$\alpha \to \alpha \mid \alpha = \tau$$

Suppose we have an identity function, which uses its argument as a  $\boldsymbol{\tau}$ 

$$\alpha \to \alpha \mid \alpha = \tau$$
$$\equiv^{\forall} \tau \to \tau$$

Suppose we have an identity function, which uses its argument as a  $\boldsymbol{\tau}$ 

$$\alpha \to \alpha \mid \alpha = \tau$$
$$\equiv^{\forall} \tau \to \tau$$

The substitution  $[\tau/\alpha]$  solves the constraint  $\alpha = \tau$ 

"solves?"

What does it mean to **solve** a constraint?

1.  $[\tau/\alpha]$  trivialises the constraint  $\alpha=\tau$  (it is a *unifier*), and all other unifiers are an instance of it (it is a *most general unifier*)

"solves?"

What does it mean to **solve** a constraint?

- 1.  $[\tau/\alpha]$  trivialises the constraint  $\alpha=\tau$  (it is a *unifier*), and all other unifiers are an instance of it (it is a *most general unifier*)
- 2. For any type  $\tau'$ , the following sets agree: the instances of  $\tau'$ , subject to  $\alpha = \tau$  the instances of  $[\tau/\alpha]\tau'$

Suppose we have an identity function, which uses its argument as a  $\tau^-$ .

$$\alpha \to \alpha \mid \alpha \le \tau^-$$

Suppose we have an identity function, which uses its argument as a  $\tau^-$ .

$$\alpha \to \alpha \mid \alpha \le \tau^{-}$$

$$\equiv^{\forall} (\alpha \sqcap \tau^{-}) \to (\alpha \sqcap \tau^{-})$$

Suppose we have an identity function, which uses its argument as a  $\tau^-$ .

$$\alpha \to \alpha \mid \alpha \le \tau^{-}$$

$$\equiv^{\forall} (\alpha \sqcap \tau^{-}) \to (\alpha \sqcap \tau^{-})$$

$$\equiv^{\forall} (\alpha \sqcap \tau^{-}) \to \alpha$$

Suppose we have an identity function, which uses its argument as a  $\tau^-$ .

$$\alpha \to \alpha \mid \alpha \le \tau^{-}$$

$$\equiv^{\forall} (\alpha \sqcap \tau^{-}) \to (\alpha \sqcap \tau^{-})$$

$$\equiv^{\forall} (\alpha \sqcap \tau^{-}) \to \alpha$$

The bisubstitution  $[\alpha \sqcap \tau^-/\alpha^-]$  solves  $\alpha \leq \tau^-$ 

#### Decomposing constraints

We only need to decompose constraints of the form  $\tau^+ \leq \tau^-$ .

$$\tau_1 \sqcup \tau_2 \le \tau_3 \quad \equiv \quad \tau_1 \le \tau_3, \ \tau_2 \le \tau_3 
\tau_1 \le \tau_2 \sqcap \tau_3 \quad \equiv \quad \tau_1 \le \tau_2, \ \tau_1 \le \tau_3$$

Thanks to the input/output type distinction, the hard cases of  $\tau_1 \sqcap \tau_2 \leq \tau_3$  and  $\tau_1 \leq \tau_2 \sqcup \tau_3$  can never come up.

## Combining solutions

We solve a system of multiple constraints  $C_1$ ,  $C_2$  by:

- ▶ Solving  $C_1$ , giving a bisubstitution  $\xi$
- Applying that to  $C_2$
- ▶ Solving  $\xi C_2$ , giving a bisubstitution  $\zeta$

Then  $\xi \circ \zeta$  solves the system  $C_1$ ,  $C_2$ .

## Putting it all together

biunify (C) takes a set of constraints C, and produces a bisubstitution solving them.

$$\begin{aligned} \operatorname{biunify}(\emptyset) &= [] \\ \operatorname{biunify}(\alpha \leq \alpha, C) &= \operatorname{biunify}(C) \\ \operatorname{biunify}(\alpha \leq \tau, C) &= \operatorname{biunify}(\theta_{\alpha \leq \tau} H; \ \theta_{\alpha \leq \tau} \ C) \circ \theta_{\alpha \leq \tau} \\ \operatorname{biunify}(\tau \leq \alpha, C) &= \operatorname{biunify}(\theta_{\tau \leq \alpha} H; \ \theta_{\tau \leq \alpha} \ C) \circ \theta_{\tau \leq \alpha} \\ \operatorname{biunify}(c, C) &= \operatorname{biunify}(\operatorname{decompose}(c), C) \end{aligned}$$

### Putting it all together

biunify (C) takes a set of constraints C, and produces a bisubstitution solving them.

$$\begin{aligned} \operatorname{biunify}(\emptyset) &= [] \\ \operatorname{biunify}(\alpha \leq \alpha, C) &= \operatorname{biunify}(C) \\ \operatorname{biunify}(\alpha \leq \tau, C) &= \operatorname{biunify}(\theta_{\alpha \leq \tau} H; \ \theta_{\alpha \leq \tau} \ C) \circ \theta_{\alpha \leq \tau} \\ \operatorname{biunify}(\tau \leq \alpha, C) &= \operatorname{biunify}(\theta_{\tau \leq \alpha} H; \ \theta_{\tau \leq \alpha} \ C) \circ \theta_{\tau \leq \alpha} \\ \operatorname{biunify}(c, C) &= \operatorname{biunify}(\operatorname{decompose}(c), C) \end{aligned}$$

Replace the  $\leq$  with = and we have Martelli and Montanari's unification algorithm.

## Summary

MLsub infers types by walking the syntax of the program, but must deal with subtyping constraints rather than just equalities. Thanks to:

- algebraically well-behaved types
- ightharpoonup polar types, restricting occurrences of  $\sqcup$  and  $\sqcap$
- a careful definition of "solves"

the biunify algorithm can always handle these constraints, producing a principal type.

Questions?

http://www.cl.cam.ac.uk/~sd601/mlsub stephen.dolan@cl.cam.ac.uk

#### Mutable references

References are generally considered "invariant".

Instead, consider ref a two-argument constructor

$$(\alpha,\beta)$$
 ref

with operations:

make: 
$$(\alpha, \alpha)$$
 ref get:  $(\bot, \beta)$  ref  $\to \beta$  set:  $(\alpha, \top)$  ref  $\to \alpha \to \text{unit}$