

# *Type inference with Subtypes (Fuh and Mishra)*

*Advanced language implementation  
and language-based security, Fall 2003*

Henning Niss

IT University of Copenhagen

# Principal types

In the simply-typed lambda calculus without subtyping (and polymorphism) each expression  $e$  has a *principal type*  $\tau$  from which all other types for  $e$  may be obtained by instantiation.

Example: the principal type of the identity function  $I \equiv \lambda x.x$  is  $\alpha \rightarrow \alpha$ .

Now consider the case with subtyping.

Example: Assume  $\text{int} <: \text{real}$ . The principal type for  $\lambda x.x$  has to be of the form  $\alpha \rightarrow \alpha$ .

However, the type  $\text{int} \rightarrow \text{real}$  for  $\lambda x.x$  is not an instance of  $\alpha \rightarrow \alpha$ .

## Principal types — cont'd

Even, if we say that a principal type is a type from which all other types can be obtained by instantiation and promotion, we still have problems.

Example: the type `int → real` is a supertype of an inst. of  $\alpha \rightarrow \alpha$ .

Example: a potential principal type for  $twice \equiv \lambda f.\lambda x.f(fx)$  is  $(\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$ .

One type  $twice$  has is `(real → int) → (real → int)`.

However, there is no type  $\tau$  such that  $\tau$  is an instance of  $(\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$ , *and*  $\tau <: \text{code}(\text{real} \rightarrow \text{int}) \rightarrow \text{code}(\text{real} \rightarrow \text{int})$ .

# Principal types

Issue: the type expression alone does not carry enough information to represent all types of an expression.

Instead look at a pair  $(C, \tau)$  consisting of *coercion set*  $C$  and a type expression  $\tau$ .

A *coercion* is a string  $\tau <: \tau'$ .

The intention is that an instance of such a pair can be obtained by giving a substitution that satisfies all coercions in  $C$  and apply that substitution to  $\tau$ .

There is an algorithm, `TYPE`, that infers principal types.

Example: for the identity function we would compute the pair  $(C, \gamma)$  where  $C = \{\alpha \rightarrow \beta <: \gamma, \alpha <: \beta\}$ .

Then, `int`  $\rightarrow$  `real` is indeed an instance of the principal type  $(C, \gamma)$  for  $I$ .

# Structural and non-structural subtyping

We have seen examples of so-called *structural subtyping*.

$$\frac{\tau'_2 <: \tau'_1 \quad \tau''_1 <: \tau''_2}{\tau'_1 \rightarrow \tau''_1 <: \tau'_2 \rightarrow \tau''_2} \quad \frac{\tau'_1 <: \tau'_2 \quad \tau''_1 <: \tau''_2}{\tau'_1 \times \tau''_1 <: \tau'_2 \times \tau''_2}$$

Only structurally similar types can be compared.

In *non-structural subtyping* that is no longer required.

Example, we might have assumptions allowing us to conclude  $\alpha <: \alpha \rightarrow \alpha$ .

# *General subtyping*

# Valid coercions and entailment

Assume given a relation *valid* on  $\text{TYPE} \times \text{TYPE}$  satisfying

- it is reflexive;
- it is transitive;
- it is closed under substitution.

Define the *entailment* relation  $\Vdash$  as follows

$$\tau <: \tau' \Vdash \sigma <: \sigma' \quad \text{iff} \quad S(\tau) <: S(\tau') \text{ valid implies } S(\sigma) <: S(\sigma') \text{ valid}$$

Lift  $\Vdash$  to coercion sets by interpreting a coercion set as a conjunction of the individual coercions.

# A subtype system for lambda calculus

$$\frac{}{\Gamma; C \vdash x : \Gamma(x)}$$

VAR

$$\frac{\Gamma, x : \tau; C \vdash e : \tau'}{\Gamma; C \vdash \lambda x. e : \tau \rightarrow \tau'}$$

ABS

$$\frac{\Gamma; C \vdash e : \tau' \rightarrow \tau \quad \Gamma; C \vdash e : \tau'}{\Gamma; C \vdash ee' : \tau}$$

APP

$$\frac{\Gamma; C \vdash e : \tau \quad C \Vdash \tau <: \tau'}{\Gamma; C \vdash e : \tau'}$$

COERCE

# Instances

A typing  $\Gamma'; C' \vdash e : \tau'$  is an instance of a typing  $\Gamma; C \vdash e : \tau$  iff there exists a substitution  $S$  such that

- $\tau' = S(\tau)$ ;
- $\Gamma' |_{fv(e)} = S(\Gamma) |_{fv(e)}$ ;
- $C' \Vdash S(C)$ .

**Lemma 1 (Substitution)** *If  $\Gamma; C \vdash e : \tau$  is typing, then so is any instance  $\Gamma'; C' \vdash e : \tau'$ .*

# Algorithm TYPE

```
TYPE( $\Gamma, e$ )  (TYPE : TYPEASSIGN  $\times$  EXP  $\rightarrow$  COERCIONSET  $\times$  TYPE)
=  C  $\leftarrow$   $\emptyset$ ;  $\alpha$  fresh; G  $\leftarrow$   $\{(\Gamma, e, \alpha)\}$ ;
   while G is not empty do
     choose any g from G;
     case g of
       ( $\Gamma, \lambda x.e, \tau$ ):
         ( $\alpha_x, \alpha_{\lambda x.e}$ ) fresh;
         C  $\leftarrow$  C  $\cup$   $\{\alpha_x \rightarrow \alpha_{\lambda x.e} <: \tau\}$ ;
         G  $\leftarrow$  (G -  $\{g\}$ )  $\cup$   $\{(\Gamma; x : \alpha_x, e, \alpha_{\lambda x.e})\}$ ;
       ( $\Gamma, ee', \tau$ ):
         ( $\alpha_{ee'}, \alpha_{e'}$ ) fresh;
         C  $\leftarrow$  C  $\cup$   $\{\alpha_{ee'} <: \tau\}$ ;
         G  $\leftarrow$  (G -  $\{g\}$ )  $\cup$   $\{(\Gamma, e, \alpha_{e'} \rightarrow \alpha_{ee'}), (\Gamma, e', \alpha_{e'})\}$ ;
       ( $\Gamma, x, \tau$ ):
         C  $\leftarrow$  C  $\cup$   $\{\Gamma(x) <: \tau\}$ ;
         G  $\leftarrow$  G -  $\{g\}$ 
     esac
   done
   return (C,  $\alpha$ )
```

# Example

Consider the expression  $e = \lambda f. \lambda x. fx$  and the type assignment  $\Gamma = \emptyset$ .  
Algorithm TYPE proceeds as follows:

# Properties of TYPE

---

**Theorem 1** TYPE *is sound and (syntactically) complete.*

# Well-typings

Not all typings are interesting!

Example:  $\text{succ} : \text{int} \rightarrow \text{int}; \{\text{real} <: \text{int}\} \vdash \text{succ } 1.5 : \text{int}.$

Idea: look only at a special form of typings.

A coercion set  $C$  is *consistent* if there exists a substitution  $S$  such that  $S(C)$  is valid.

A *well-typing* is a typing  $\Gamma; C \vdash e : \tau$  where  $C$  is consistent.

**Lemma 2** *If  $\Gamma; C \vdash e : \tau$  is a well-typing, then so is every instance  $\Gamma'; C' \vdash e : \tau'$  as long as  $C'$  is consistent.*

# Inferring well-typings

Algorithm WTYPE computes well-typings:

```
let  $(C, \tau) = \text{TYPE}(\Gamma, e)$   
in if  $C$  is consistent then  $C; \Gamma \vdash e : \tau$   
    else fail
```

**Theorem 2** WTYPE is sound and complete.

# *Structural subtyping*

# Subtype logic

Previously, we just postulated a reflexive, transitive, and substitution-closed relation  $\Vdash$ .

Now, define the relation as follows.

$$\frac{}{C \Vdash \text{int} <: \text{real}} \quad \frac{}{C \cup \{\tau <: \sigma\} \Vdash \tau <: \sigma}$$
$$\frac{}{C \Vdash \tau <: \tau} \quad \frac{C \Vdash \tau_1 <: \tau_2 \quad C \Vdash \tau_2 <: \tau_3}{C \Vdash \tau_1 <: \tau_3}$$
$$\frac{C \Vdash \tau'_1 <: \tau_1 \quad C \Vdash \tau_2 <: \tau'_2}{C \Vdash \tau_1 \rightarrow \tau_2 <: \tau'_1 \rightarrow \tau'_2}$$

We say that  $C \Vdash \tau <: \sigma$  is valid if it is derivable by the above rules.

Coercion  $\tau <: \sigma$  is valid iff  $\emptyset \Vdash \tau <: \sigma$ .

# Information content

Example: under a structural subtyping regime, any valid instance of the typing  $\Gamma; \{\alpha <: \beta \rightarrow \gamma\} \vdash e : \alpha$  must instantiate  $\alpha$  to a function type.

Thus the set of valid instances of  $\Gamma; \{\alpha <: \beta \rightarrow \gamma\} \vdash e : \alpha$  and  $\Gamma; \{\alpha' \rightarrow \alpha'' <: \beta \rightarrow \gamma\} \vdash e : \alpha' \rightarrow \alpha''$  are the same.

On the other hand, the set of valid instances of  $\Gamma; \{\alpha' \rightarrow (\alpha'' \rightarrow \alpha''') <: \beta \rightarrow (\gamma' \rightarrow \gamma'')\} \vdash e : \alpha' \rightarrow (\alpha'' \rightarrow \alpha''')$  is smaller than this (intuitively, we have “over-instantiated”).

We would like to find a well-typing  $\Gamma_*; C_* \vdash e : \alpha_*$  such that any instance of  $\Gamma; \{\alpha <: \beta \rightarrow \gamma\} \vdash e : \alpha$  is also an instance of this well-typing, and such that  $\Gamma_*$ ,  $C_*$ ,  $\alpha_*$  provide as simple as possible, but also as “structural” as possible information.

# Matchings

Define relation *match* by

- $match(\tau, \tau')$  is true if both  $\tau$  and  $\tau'$  are atomic;
- $match(\tau_1 \rightarrow \tau_2, \tau'_1 \rightarrow \tau'_2)$  if  $match(\tau_1, \tau'_1)$  and  $match(\tau_2, \tau'_2)$ .

A coercion  $\tau <: \tau'$  is *matching* if  $match(\tau, \tau')$  is true.

If  $C$  is a valid coercion set, then every coercion in  $C$  must be matching.

In other words, instantiating a coercion set to be matching preserves the set of valid instances.

Example: the set of valid instances of  $\Gamma; \{\alpha <: \beta \rightarrow \gamma\} \vdash e : \alpha$  and  $\Gamma; \{\alpha' \rightarrow \alpha'' <: \beta \rightarrow \gamma\} \vdash e : \alpha' \rightarrow \alpha''$  are the same.

# Minimal matching instances

**Theorem 3** *Given a well-typing  $\Gamma; C \vdash e : \tau$ , there exists a well-typing  $\Gamma_*; C_* \vdash e : \tau_*$  such that*

- $\Gamma_*; C_* \vdash e : \tau_*$  is matching instance of  $\Gamma; C \vdash e : \tau$ ;
- $\Gamma; C \vdash e : \tau$  and  $\Gamma_*; C_* \vdash e : \tau_*$  have the same information content;
- if  $\Gamma'; C' \vdash e : \tau'$  is another matching instance of  $\Gamma; C \vdash e : \tau$ , then it is also an instance of  $\Gamma_*; C_* \vdash e : \tau_*$ .

We say that  $\Gamma_*; C_* \vdash e : \tau_*$  is the minimal matching instance of  $\Gamma; C \vdash e : \tau$ .

If  $\Gamma'; C' \vdash e : \tau'$  is an instance of  $\Gamma; C \vdash e : \tau$ , then it is also an instance of  $\Gamma_*; C_* \vdash e : \tau_*$ .

# Example: minimal matching instance

Consider,  $\lambda f.\lambda x.fx$  again, and the coercion set

$$C = \left\{ \begin{array}{l} \alpha_f \rightarrow \alpha_{\lambda x.fx} <: \alpha, \\ \alpha_x \rightarrow \alpha_{fx} <: \alpha_{\lambda x.fx}, \\ \alpha_2 <: \alpha_{fx}, \\ \alpha_f <: \alpha_1 \rightarrow \alpha_2, \\ \alpha_x <: \alpha_1 \end{array} \right\}$$

and resulting type  $\alpha$ .

The minimal matching instance would be

$$C_* = \left\{ \begin{array}{l} (\alpha'_f \rightarrow \alpha''_f) \rightarrow (\alpha'_{\lambda x.fx} \rightarrow \alpha''_{\lambda x.fx}) <: (\beta_1 \rightarrow \beta_2) \rightarrow (\beta_3 \rightarrow \beta_4), \\ \alpha_x \rightarrow \alpha_{fx} <: \alpha'_{\lambda x.fx} \rightarrow \alpha''_{\lambda x.fx}, \\ \alpha_2 <: \alpha_{fx}, \\ \alpha'_f \rightarrow \alpha''_f <: \alpha_1 \rightarrow \alpha_2, \\ \alpha_x <: \alpha_1 \end{array} \right\}$$

with resulting type  $\alpha_* = (\beta_1 \rightarrow \beta_2) \rightarrow (\beta_3 \rightarrow \beta_4)$ .

# Simplifying coercion sets

Example: consider coercion sets  $C_1 = \{\alpha \rightarrow \alpha' <: \beta \rightarrow \beta'\}$  and  $C_2 = \{\beta <: \alpha, \alpha' <: \beta'\}$ .

It is easy to verify that  $C_1 \Vdash C_2$  and  $C_2 \Vdash C_1$ ; i.e., the coercion sets are equivalent.

Intuitively,  $C_2$  is preferable to  $C_1$  as it contains less redundant information.

A coercion  $\tau <: \tau'$  is *atomic* if both  $\tau$  and  $\tau'$  are atomic (either, type variables or type constants).

Algorithm SIMPLIFY computes maximally simplified coercion sets consisting only of atomic coercions:

- $\text{SIMPLIFY}(C) = C$  if all coercions in  $C$  are atomic;
- $\text{SIMPLIFY}(C \cup \{\tau \rightarrow \tau' <: \sigma \rightarrow \sigma'\}) = \text{SIMPLIFY}(C \cup \{\sigma <: \tau, \tau' <: \sigma'\})$ .

# Example: simplifying coercion sets

The coercion set

$$C_* = \left\{ \begin{array}{l} (\alpha'_f \rightarrow \alpha''_f) \rightarrow (\alpha'_{\lambda x.fx} \rightarrow \alpha''_{\lambda x.fx}) <: (\beta_1 \rightarrow \beta_2) \rightarrow (\beta_3 \rightarrow \beta_4), \\ \alpha_x \rightarrow \alpha_{fx} <: \alpha'_{\lambda x.fx} \rightarrow \alpha''_{\lambda x.fx}, \\ \alpha_2 <: \alpha_{fx}, \\ \alpha'_f \rightarrow \alpha''_f <: \alpha_1 \rightarrow \alpha_2, \\ \alpha_x <: \alpha_1 \end{array} \right\}$$

can be simplified to

$$C_* = \left\{ \begin{array}{l} \alpha'_f <: \beta_1, \beta_2 <: \alpha''_f, \\ \beta_3 <: \alpha'_{\lambda x.fx}, \alpha''_{\lambda x.fx} <: \beta_4, \alpha'_{\lambda x.fx} <: \alpha_x, \alpha_{fx} <: \alpha''_{\lambda x.fx}, \\ \alpha_2 <: \alpha_{fx}, \\ \alpha_1 <: \alpha'_f, \alpha''_f <: \alpha_2 \\ \alpha_x <: \alpha_1 \end{array} \right\}$$

In reality we are then only interested in  $\{\beta_2 <: \beta_4, \beta_3 <: \beta_1\}$

# Algorithm WTYPE

---

```
WTYPE( $\Gamma, e$ )
=   let ( $C, \alpha$ ) = TYPE( $\Gamma, e$ )
   in if  $C$  is consistent
      then let  $\Gamma_*; C_* \vdash e : \alpha_*$  be the minimal matching instance
         in  $\Gamma_*; \text{SIMPLIFY}(C_*) \vdash e : \alpha_*$ 
      else fail
```

# Algorithm MATCH

```
MATCH( $C_0$ )
= ( $C, S, M$ )  $\leftarrow$  ( $C_0, Id, \{(a, a) | a \in C_0\}$ );
  while  $C \neq \emptyset$  do
    choose any  $c$  from  $C$ ;
    case  $c$  of
       $\tau_1 \rightarrow \tau'_1 <: \tau_2 \rightarrow \tau'_2$ : perform Decomposition
       $a_1 <: a_2$ : perform Atomic elimination
       $\alpha <: \tau$  or  $\tau <: \alpha$ : perform Expansion
    esac
  done
  return  $S$ 
```

# Properties of MATCH

---

**Theorem 4** *MATCH always terminates.*

**Theorem 5** *If  $C_0$  is not matchable then  $\text{MATCH}(C_0)$  fails, otherwise  $\text{MATCH}(C_0)$  returns a substitution  $S$  such that  $S(C_0)$  is the minimal matching instance of  $C_0$ .*