Boxy types

Inference for higher-rank types and impredicativity

Dimitrios Vytiniotis[1] Stephanie Weirich[1]
Simon Peyton Jones[2]

[1] Computer and Information Science Department
University of Pennsylvania

[2] Microsoft Research

Presentation for ICFP 2006—Portland, Oregon

Trends in functional programming

. . . generalized algebraic datatypes [e.g. Weirich's talk in ICFP'06], polymorphic recursion, higher-rank and impredicative polymorphism, type-level lambdas, existential types, dependent types . . .

Trends in functional programming

. . . generalized algebraic datatypes [e.g. Weirich's talk in ICFP'06], polymorphic recursion, higher-rank and impredicative polymorphism, type-level lambdas, existential types, dependent types . . .

- Ideas originating in explicitly typed languages
- Full type inference hard: need mixture of checking and inference

Trends in functional programming

…generalized algebraic datatypes [e.g. Weirich's talk in ICFP'06], polymorphic recursion, higher-rank and impredicative polymorphism, type-level lambdas, existential types, dependent types …

- Ideas originating in explicitly typed languages
- Full type inference hard: need mixture of checking and inference

Higher-rank types

A type of **higher-rank**: ∀-quantifiers nested to the left of arrows

Higher-rank types can be indispensable [Shan, 2004]:

```
-- generic map from SYB
gmapT :: forall a. Term a => (forall b. Term b => b -> b) -> a -> a


-- encapsulated state monad
runST :: forall a. (forall s. ST s a) -> a
```

Impredicative instantiation

Impredicative instantiation:

Quantified type variables can be instantiated with arbitrary types

Create and use data structures that hold polymorphic values:

```
head :: forall a. [a] -> a
ids  :: [forall c. c -> c]
f = head ids
```

$head\ ids$ requires instantiation of a with $\forall c.c \to c$

Type inference

There is need for type inference—but not an easy problem

```
f g = (g 3, g True)
```

Which type do we infer for f?

$$\vdash f : (\forall a.a \to Int) \to (Int, Int)$$

$$\vdash f : (\forall a.a \to a) \to (Int, Bool)$$

$$\vdash f : \; \ldots\text{other?}$$

- No "best" type for f. No **modularity**
- Worse: Typability for System-F **undecidable** [Wells, 1994]

Modest solutions for higher-rank types

Putting type annotations to work:

1. **Annotated** functions take polymorphic args [Odersky and Läufer, 1996]

```
f (g::(forall a.a->a)) = (g 3, g True)
```

2. Annotations **not exactly on** the functions; bidirectional propagation [Peyton Jones *et al.*, 2003, 2005][Rémy, 2005]

```
f :: (forall a.a->Int)->(Int,Int)
f = \g -> (g 3, g True)
```

...but no impredicative instantiation

Haskell + higher-rank types + impredicativity $\approx$ programming in System-F

Boxy types

1. Hindley-Milner style type inference
   - **plus** impredicative and higher-rank polymorphism!

2. Follow "putting type annotations to work" ideas

Design principles

1. Only include System-F types; unlike MLF [Le Botlan and Rémy, 2003]
2. Simple extension to existing type inference algorithms
   - Never guess polymorphism; propagate type annotations instead
   - Single-pass; unlike Rémy's stratified type inference [Rémy, 2005]
3. Reach all of System-F via addition of appropriate annotations
4. Straightforward semantics-preserving compilation to System-F

Boxy types, concretely

Take System-F types and draw **boxes** around them, e.g.

$$\boxed{\forall a . a \to a} \to Int \to \boxed{Bool}$$

$$\boxed{\forall a . a \to a \to a}$$

Judgements of the form:

$$\langle \text{environment} \rangle \vdash \langle \text{expression} \rangle : \langle \text{boxy-type} \rangle$$

- Boxes represent **inferred** information
  - Output parameters to the type checker
- Box-free parts represent **known information**
  - Input parameters to the type checker
  - Known info originating in type annotations

Boxy types for information propagation

No special semantics for boxes

- Only there to restrict **which typing derivations are valid**

$$\vdash \lambda g.(g\ 3, g\ \mathit{True}) : \underbrace{(\forall a.a \to a) \to (\mathit{Int}, \mathit{Bool})}_{\text{known info}}$$

Boxy types for information propagation

No special semantics for boxes

- Only there to restrict **which typing derivations are valid**

$$\vdash \lambda g.(g\ 3, g\ \mathit{True}) : \underbrace{(\forall a.a \to a) \to (\mathit{Int}, \mathit{Bool})}_{\text{known info}}$$

$$\vdash \lambda g.(g\ 3, g\ \mathit{True}) : \underbrace{(\forall a.a \to a)}_{\text{known info}} \to \boxed{(\mathit{Int}, \mathit{Bool})}$$

Boxy types for information propagation

No special semantics for boxes

- Only there to restrict **which typing derivations are valid**

$$\vdash \lambda g.(g\ 3, g\ \mathit{True}) : \underbrace{(\forall a.a \to a) \to (\mathit{Int}, \mathit{Bool})}_{\text{known info}}$$

$$\vdash \lambda g.(g\ 3, g\ \mathit{True}) : \underbrace{(\forall a.a \to a)}_{\text{known info}} \to \boxed{(\mathit{Int}, \mathit{Bool})}$$

$$\nvdash \lambda g.(g\ 3, g\ \mathit{True}) : \underbrace{\boxed{\forall a.a \to a} \to \boxed{(\mathit{Int}, \mathit{Bool})}}_{\text{both boxes inferred!}} \quad \Leftarrow \text{Can't do that!}$$

Boxy types for information propagation

No special semantics for boxes

- Only there to restrict **which typing derivations are valid**

$$\vdash \lambda g.(g\ 3, g\ \mathit{True}) : \underbrace{(\forall a.a \to a) \to (\mathit{Int}, \mathit{Bool})}_{\text{known info}}$$

$$\vdash \lambda g.(g\ 3, g\ \mathit{True}) : \underbrace{(\forall a.a \to a)}_{\text{known info}} \to \boxed{(\mathit{Int}, \mathit{Bool})}$$

$$\nvdash \lambda g.(g\ 3, g\ \mathit{True}) : \underbrace{\boxed{\forall a.a \to a} \to \boxed{(\mathit{Int}, \mathit{Bool})}}_{\text{both boxes inferred!}} \quad \Leftarrow \text{Can't do that!}$$

$$\vdash \lambda \underbrace{(g :: \forall a.a \to a)}_{\text{provides info}}.(g\ 3, g\ \mathit{True}) : \boxed{\forall a.a \to a} \to \boxed{(\mathit{Int}, \mathit{Bool})}$$

Filling in boxes: Never guess polymorphism

- Boxes filled in with info available **locally**

$$\vdash \lambda(g :: \forall a.a \to a).(g\ 3, g\ \mathit{True}) : \boxed{\forall a.a \to a} \to (\mathit{Int}, \mathit{Bool})$$

Known info not only annotations, but also box-free type parts

$$\vdash \lambda x.x : [\forall a.a \to a] \to \underbrace{\boxed{[\forall a.a \to a]}}_{\text{filled by known info}}$$

Filling in boxes: Never guess polymorphism

- Boxes filled in with info available **locally**

$$\vdash \lambda(g :: \forall a.a \to a).(g\ 3, g\ \mathit{True}) : \boxed{\forall a.a \to a} \to (\mathit{Int}, \mathit{Bool})$$

Known info not only annotations, but also box-free type parts

$$\vdash \lambda x.x : [\forall a.a \to a] \to \underbrace{\boxed{[\forall a.a \to a]}}_{\text{filled by known info}}$$

- No info available locally: contents **monomorphic**

$$\not\vdash \lambda x.x : \boxed{(\forall a.a \to a)} \to \boxed{(\forall a.a \to a)}$$

Instead:

$$\vdash \lambda x.x : \boxed{\tau} \to \boxed{\tau} \quad \text{(for any monomorphic } \tau\text{)}$$

Impredicative instantiation

Working example $\lambda g.(g\ 3, g\ \mathit{True})$ requires a higher-rank type

Q: What about **impredicative instantiation**?

A: Same principle! Never "guess" polymorphism!

Impredicative instantiation

Example:

$$f{:}\forall a.a \to a \in \Gamma$$

Instantiate a to derive:

$$\Gamma \vdash f : \boxed{Int} \to \boxed{Int}$$

But **never guess arbitrary polymorphic instantiations**:

$$\Gamma \nvdash f : \underbrace{\boxed{(\forall a.a \to a)} \to \boxed{(\forall a.a \to a)}}_{\text{no known info to determine instantiation}}$$

Instead, **check arbitrary polymorphic instantiations**:

$$\Gamma \vdash f : \underbrace{(\forall a.a \to a) \to (\forall a.a \to a)}_{\text{known info determines instantiation!}}$$

Boxes, formally

$$
\begin{array}{rcl}
\sigma & ::= & \forall\overline{a}.\rho \\
\rho & ::= & \sigma \to \sigma \mid \tau \\
\tau & ::= & a \mid \tau \to \tau
\end{array}
\quad\Bigg|\quad
\begin{array}{rcl}
\sigma' & ::= & \forall\overline{a}.\rho' \mid \boxed{\sigma} \\
\rho' & ::= & \sigma' \to \sigma' \mid \boxed{\rho} \mid \tau
\end{array}
$$

- Constraints:
  - No nested boxes: guess$^2 \equiv$ guess
  - No boxes in environments: search-free, single-pass, more in paper
  - No quantified variables free inside boxes: what would that mean?

Call τ **monotypes**, rest **polytypes**

Typing derivations

The high-level specification of an inference algorithm:

1. Rules directed by syntax of terms **and** types
2. Close to Haskell syntax-directed rules:
   - Instantiation occurs at variables
   - Generalization at (unannotated) `let` expressions
3. Intuition: When not sure, use a box!

$$\frac{\Gamma \vdash t : \boxed{\sigma} \to \rho' \quad \Gamma \vdash u : \sigma}{\Gamma \vdash t\ u : \rho'}\ \textsc{App}$$

- Check t using known info from ρ' and inferring $\boxed{\sigma}$ for argument
- Check u against σ

More typing

- Type annotations result in fully-checkable types

$$\frac{\Gamma \vdash u : \sigma \quad \Gamma, x{:}\sigma \vdash t : \rho'}{\Gamma \vdash \texttt{let x} :: \sigma = \texttt{u in t} : \rho'} \ \textsc{SigLet}$$

More typing

- Type annotations result in fully-checkable types

$$\frac{\Gamma \vdash u : \sigma \quad \Gamma, x{:}\sigma \vdash t : \rho'}{\Gamma \vdash \texttt{let x} :: \sigma = \texttt{u in t} : \rho'} \ \text{SIGLET}$$

- No information; fully inferred

$$\frac{\begin{array}{c}\Gamma \vdash u : \boxed{\rho} \\ \overline{a} = \mathit{ftv}(\rho) - \mathit{ftv}(\Gamma) \\ \Gamma, x{:}\forall\overline{a}.\rho \vdash t : \rho'\end{array}}{\Gamma \vdash \texttt{let}\ x = u\ \texttt{in}\ t : \rho'} \ \text{LET}$$

Shape of types drives typing

$$\frac{\Gamma \vdash (\lambda x.t) : \boxed{\sigma_1} \to \boxed{\sigma_2}}{\Gamma \vdash (\lambda x.t) : \boxed{\sigma_1 \to \sigma_2}}\ \textsc{abs}1 \qquad \frac{\Gamma \vdash t : \rho' \quad \overline{a} \cap \mathit{ftv}(\Gamma) = \emptyset}{\Gamma \vdash t : \forall \overline{a}.\rho'}\ \textsc{skol}$$

- Types $\boxed{\sigma_1} \to \boxed{\sigma_2}$ and $\boxed{\sigma_1 \to \sigma_2}$ are effectively interchangeable
- SKOL "skolemizes" quantified variables

Instantiation

As in Haskell:

$$\frac{\vdash \sigma \leq \rho' \quad x{:}\sigma \in \Gamma}{\Gamma \vdash x : \rho'}\ \mathrm{VAR}$$

($\leq$): **subsumption** relation. Read σ is "more polymorphic than" ρ'

Let $\sigma_{id} = \forall a.a \to a$. When $f{:}\sigma_{id} \in \Gamma$ we wish:

- $\Gamma \vdash f : \boxed{Int} \to \boxed{Int}$ therefore $\vdash \forall a.a \to a \leq \boxed{Int} \to \boxed{Int}$
- $\Gamma \not\vdash f : \boxed{\sigma_{id}} \to \boxed{\sigma_{id}}$ therefore $\not\vdash \forall a.a \to a \leq \boxed{\sigma_{id}} \to \boxed{\sigma_{id}}$
- $\Gamma \vdash f : \sigma_{id} \to \sigma_{id}$ therefore $\vdash \forall a.a \to a \leq \sigma_{id} \to \sigma_{id}$

Operation of subsumption

- Instantiation of left-hand side variables with **boxes**

$$\frac{\vdash [a \mapsto \boxed{\sigma}]\rho_1' \leq \rho_2'}{\vdash \forall a.\rho_1' \leq \rho_2'}\ \textsc{Spec}$$

- Filling-in of boxes from both sides
  - Box-meets-box situation: Force to monotypes!

Operation of subsumption, continued

$$\nvdash \forall a.a \to a \leq \boxed{\sigma_{id}} \to \boxed{\sigma_{id}}$$

$$\hookleftarrow \quad \vdash \boxed{?} \to \boxed{?} \leq \boxed{\sigma_{id}} \to \boxed{\sigma_{id}}$$

$$\hookleftarrow \quad \vdash \underbrace{\boxed{?} \leq \boxed{\sigma_{id}}}_{\text{result types}} \wedge \quad \vdash \underbrace{\boxed{?} \sim \boxed{\sigma_{id}}}_{\text{invariant argument types}}$$

$\hookleftarrow$ Fail: σ_{id} not monotype!

Theorem: $\nvdash \boxed{\forall a.\sigma} \leq \boxed{\forall a.\sigma}$ and $\vdash \boxed{\tau} \leq \boxed{\tau}$

($\Longrightarrow$ hence also $\vdash \forall a.a \to a \leq \boxed{Int} \to \boxed{Int}$)

Operation of subsumption, continued

On the other hand:

$$
\begin{array}{rl}
 & \vdash \forall a.a \to a \leq \sigma_{id} \to \sigma_{id} \\
\hookleftarrow & \vdash \boxed{?} \to \boxed{?} \leq \sigma_{id} \to \sigma_{id} \\
\hookleftarrow & \underbrace{\vdash \boxed{?} \leq \sigma_{id}}_{\text{result types}} \wedge \underbrace{\vdash \boxed{?} \sim \sigma_{id}}_{\text{argument types}} \\
\hookleftarrow & ? = \sigma_{id}
\end{array}
$$

Theorem: $\vdash \forall a.\sigma \leq [a \mapsto \sigma_a]\sigma$

Abstractions

- One more abstraction rule:

$$\frac{\vdash \sigma_1' \sim \boxed{\sigma_1} \quad \Gamma, x{:}\sigma_1 \vdash t : \sigma_2'}{\Gamma \vdash (\lambda x.t) : \sigma_1' \to \sigma_2'}\ \textsc{abs2}$$

- $\vdash \sigma_1' \sim \boxed{\sigma_1}$ effectively forces boxes of σ_1' to monotypes

Technical properties

1. Type-safety via **straightforward** translation to System-F
2. Inference algorithm can be read-off from typing rules
3. “Principal” types for a given amount of “checked” information
4. Type system conservatively extends Hindley-Milner
5. All of System-F typeable with addition of annotations
    - placed on **polymorphic** type instantiation sites
    - but many heuristics to improve this, see paper

Future work

1. Boxy-style propagation for existential, higher-order types, dependent types
2. Practical evaluation
   - Implementation already released in GHC
3. Bridge gap between MLF and Boxy Types
   - Delaying of instantiation decisions, logical specification, etc.
4. Increase robustness under program transformations

Thank you for your attention!

Boxes are **GOOD FUN!**

Thank you for your attention!

Boxes are **GOOD FUN!**

Please read the paper for details:

`www.cis.upenn.edu/~dimitriv/boxy`

MLF

MLF [Le Botlan and Rémy,2003]:

- **Different type structure** than that of System-F
- Regain principal types and decidable type inference
- Keep expressions generalized, make use of type annotations for FCP
- Instantiation constraints on quantified type variables
- Trick: Constraints permit delaying of instantiation decisions
- Evidence translation to System-F [Leijen and Löh] but not proven computationally light

Subsumption, instance rules

$$\frac{\vdash \boxed{\sigma} \sim \sigma'}{\vdash \boxed{\sigma} \leq \sigma'}\ \textsc{sboxy} \qquad \frac{\vdash \forall \overline{a}.\rho_1' \leq \rho_2' \quad \overline{b} \notin ftv(\forall \overline{a}.\rho_1')}{\vdash \forall \overline{a}^{+}.\rho_1' \leq \forall \overline{b}.\rho_2'}\ \textsc{skol}$$

$$\frac{\vdash [\overline{a \mapsto \boxed{\sigma}}]\rho_1' \leq \rho_2'}{\vdash \forall \overline{a}.\rho_1' \leq \rho_2'}\ \textsc{spec}$$

Subumption, congruence rules

$$\frac{\vdash \sigma_1' \to \sigma_2' \leq \boxed{\sigma_3} \to \boxed{\sigma_4}}{\vdash \sigma_1' \to \sigma_2' \leq \boxed{\sigma_3 \to \sigma_4}}\ \textsc{f1} \qquad \frac{\vdash \sigma_3' \sim \sigma_1' \quad \vdash \sigma_2' \leq \sigma_4'}{\vdash \sigma_1' \to \sigma_2' \leq \sigma_3' \to \sigma_4'}\ \textsc{f2}$$

$$\frac{}{\vdash \tau \leq \tau}\ \textsc{mono}$$

- Notice function argument **invariance**. Completeness reasons.

Boxy matching (a.k.a. super-unification)

$$\vdash \sigma_1' \sim \sigma_2'$$

Intuition: Walk down the type structure and when:

Boxy matching (a.k.a. super-unification)

$$\vdash \sigma_1' \sim \sigma_2'$$

Intuition: Walk down the type structure and when:

- **a box meets a sans-box type** $\Longrightarrow$ fill in the box!

Boxy matching (a.k.a. super-unification)

$$\vdash \sigma_1' \sim \sigma_2'$$

Intuition: Walk down the type structure and when:

- **a box meets a sans-box type** $\Longrightarrow$ fill in the box!
- **a box meets a box** $\Longrightarrow$ force both to be monotypes!

Boxy matching (a.k.a. super-unification)

$$\vdash \sigma_1' \sim \sigma_2'$$

Intuition: Walk down the type structure and when:

- **a box meets a sans-box type** $\Longrightarrow$ fill in the box!
- **a box meets a box** $\Longrightarrow$ force both to be monotypes!

Examples:

$$\vdash \mathit{Int} \rightarrow \boxed{\mathit{Int}} \sim \boxed{\mathit{Int} \rightarrow \mathit{Int}}$$

Boxy matching (a.k.a. super-unification)

$$\vdash \sigma'_1 \sim \sigma'_2$$

Intuition: Walk down the type structure and when:

- **a box meets a sans-box type** $\Longrightarrow$ fill in the box!
- **a box meets a box** $\Longrightarrow$ force both to be monotypes!

Examples:

$$\vdash \mathit{Int} \to \boxed{\mathit{Int}} \sim \boxed{\mathit{Int} \to \mathit{Int}}$$
$$\not\vdash \mathit{Int} \to \boxed{\forall a.a \to a} \sim \boxed{\mathit{Int} \to \forall a.a \to a}$$

Boxy matching (a.k.a. super-unification)

$$\vdash \sigma_1' \sim \sigma_2'$$

Intuition: Walk down the type structure and when:

- **a box meets a sans-box type** $\Longrightarrow$ fill in the box!
- **a box meets a box** $\Longrightarrow$ force both to be monotypes!

Examples:

$$\vdash Int \rightarrow \boxed{Int} \sim \boxed{Int \rightarrow Int}$$
$$\not\vdash Int \rightarrow \boxed{\forall a.a \rightarrow a} \sim \boxed{Int \rightarrow \forall a.a \rightarrow a}$$
$$\vdash Int \rightarrow \forall a.a \rightarrow a \sim \boxed{Int \rightarrow \forall a.a \rightarrow a}$$

Boxy matching (a.k.a. super-unification)

$$\vdash \sigma_1' \sim \sigma_2'$$

Intuition: Walk down the type structure and when:

- **a box meets a sans-box type** $\Longrightarrow$ fill in the box!
- **a box meets a box** $\Longrightarrow$ force both to be monotypes!

Examples:

$$\vdash Int \to \boxed{Int} \sim \boxed{Int \to Int}$$
$$\not\vdash Int \to \boxed{\forall a.a \to a} \sim \boxed{Int \to \forall a.a \to a}$$
$$\vdash Int \to \forall a.a \to a \sim \boxed{Int \to \forall a.a \to a}$$
$$\vdash \boxed{a} \sim \boxed{a}$$

Boxy matching (a.k.a. super-unification)

$$\vdash \sigma_1' \sim \sigma_2'$$

Intuition: Walk down the type structure and when:

- **a box meets a sans-box type** $\Longrightarrow$ fill in the box!
- **a box meets a box** $\Longrightarrow$ force both to be monotypes!

Examples:

$$\vdash Int \to \boxed{Int} \sim \boxed{Int \to Int}$$
$$\not\vdash Int \to \boxed{\forall a.a \to a} \sim \boxed{Int \to \forall a.a \to a}$$
$$\vdash Int \to \forall a.a \to a \sim \boxed{Int \to \forall a.a \to a}$$
$$\vdash \boxed{a} \sim \boxed{a}$$
$$\not\vdash \boxed{\forall a.a \to a} \sim \boxed{\forall a.a \to a}$$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$
- $\vdash Int \to \forall a.a \to a \leq \forall a.Int \to a \to a$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$
- $\vdash Int \to \forall a.a \to a \leq \forall a.Int \to a \to a$
- $\nvdash \forall a.a \to a \leq \boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$
- $\vdash Int \to \forall a.a \to a \leq \forall a.Int \to a \to a$
- $\nvdash \forall a.a \to a \leq \boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$
- $\vdash \forall a.a \to a \leq (\forall a.a \to a) \to (\forall a.a \to a)$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$
- $\vdash Int \to \forall a.a \to a \leq \forall a.Int \to a \to a$
- $\not\vdash \forall a.a \to a \leq \boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$
- $\vdash \forall a.a \to a \leq (\forall a.a \to a) \to (\forall a.a \to a)$
- $\vdash \forall a.a \to a \leq \boxed{b} \to \boxed{b}$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$
- $\vdash Int \to \forall a.a \to a \leq \forall a.Int \to a \to a$
- $\not\vdash \forall a.a \to a \leq \boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$
- $\vdash \forall a.a \to a \leq (\forall a.a \to a) \to (\forall a.a \to a)$
- $\vdash \forall a.a \to a \leq \boxed{b} \to \boxed{b}$
- $\not\vdash \boxed{\forall a.a \to a} \leq \boxed{\forall a.a \to a}$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$
- $\vdash Int \to \forall a.a \to a \leq \forall a.Int \to a \to a$
- $\nvdash \forall a.a \to a \leq \boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$
- $\vdash \forall a.a \to a \leq (\forall a.a \to a) \to (\forall a.a \to a)$
- $\vdash \forall a.a \to a \leq \boxed{b} \to \boxed{b}$
- $\nvdash \boxed{\forall a.a \to a} \leq \boxed{\forall a.a \to a}$
- $\vdash \boxed{\tau} \leq \boxed{\tau}$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$
- $\vdash Int \to \forall a.a \to a \leq \forall a.Int \to a \to a$
- $\not\vdash \forall a.a \to a \leq \boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$
- $\vdash \forall a.a \to a \leq (\forall a.a \to a) \to (\forall a.a \to a)$
- $\vdash \forall a.a \to a \leq \boxed{b} \to \boxed{b}$
- $\not\vdash \boxed{\forall a.a \to a} \leq \boxed{\forall a.a \to a}$
- $\vdash \boxed{\tau} \leq \boxed{\tau}$
- $\vdash \forall a.a \to a \leq \boxed{Int \to Int}$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$
- $\vdash Int \to \forall a.a \to a \leq \forall a.Int \to a \to a$
- $\nvdash \forall a.a \to a \leq \boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$
- $\vdash \forall a.a \to a \leq (\forall a.a \to a) \to (\forall a.a \to a)$
- $\vdash \forall a.a \to a \leq \boxed{b} \to \boxed{b}$
- $\nvdash \boxed{\forall a.a \to a} \leq \boxed{\forall a.a \to a}$
- $\vdash \boxed{\tau} \leq \boxed{\tau}$
- $\vdash \forall a.a \to a \leq \boxed{Int \to Int}$
- $\vdash \boxed{(\forall a.a \to a) \to \forall ab.a \to b} \leq (\forall a.a \to a) \to \forall ab.a \to b$

Subsumption examples

- $\vdash (\forall ab.a \to b) \to (\forall a.a \to a) \leq (\forall ab.a \to b) \to (Int \to Int)$
- $\vdash Int \to \forall a.a \to a \leq \forall a.Int \to a \to a$
- $\nvdash \forall a.a \to a \leq \boxed{\forall a.a \to a} \to \boxed{\forall a.a \to a}$
- $\vdash \forall a.a \to a \leq (\forall a.a \to a) \to (\forall a.a \to a)$
- $\vdash \forall a.a \to a \leq \boxed{b} \to \boxed{b}$
- $\nvdash \boxed{\forall a.a \to a} \leq \boxed{\forall a.a \to a}$
- $\vdash \boxed{\tau} \leq \boxed{\tau}$
- $\vdash \forall a.a \to a \leq \boxed{Int \to Int}$
- $\vdash \boxed{(\forall a.a \to a) \to \forall ab.a \to b} \leq (\forall a.a \to a) \to \forall ab.a \to b$
- $\vdash (\forall ab.a \to b) \to \forall a.a \to a \leq \boxed{\forall ab.a \to b} \to (Int \to Int)$

Program example

$$
\begin{array}{ll}
\texttt{let} & f\ x = x \\
 & z :: [\forall a.a \to a] = [(f :: (\forall a.a \to a) \to (\forall a.a \to a))\ (\lambda x.x)] \\
\texttt{in} & ((f :: [\forall a.a \to a] \to [\forall a.a \to a])\ z)
\end{array}
$$

The safe game:

- Annotations on polymorphic instantiations will always work
- But raises questions about flow of information (look at z's type)

Note: Invariant list constructor

Program example

Infer $\boxed{a \to a}$ then unbox and generalize $f{:}\forall a.a \to a$

$$
\begin{array}{ll}
\texttt{let} & f\ x = x \\
 & z :: [\forall a.a \to a] = [(f :: (\forall a.a \to a) \to (\forall a.a \to a))\ (\lambda x.x)] \\
\texttt{in} & ((f :: [\forall a.a \to a] \to [\forall a.a \to a])\ z)
\end{array}
$$

The safe game:

- Annotations on polymorphic instantiations will always work
- But raises questions about flow of information (look at z's type)

Note: Invariant list constructor

Program example

1. $\vdash (f :: (\forall a.a \to a) \to (\forall a.a \to a))\ (\lambda x.x) : \forall a.a \to a$
2. $\vdash (\forall a.a \to a) \to (\forall a.a \to a) \leq \boxed{\forall a.a \to a} \to (a \to a)$
3. $\vdash \forall a.a \to a \leq (\forall a.a \to a) \to (\forall a.a \to a)$
4. $\vdash \lambda x.x : \forall a.a \to a$

$$
\begin{array}{ll}
\texttt{let} & f\ x = x \\
 & z :: [\forall a.a \to a] = [(f :: (\forall a.a \to a) \to (\forall a.a \to a))\ (\lambda x.x)] \\
\texttt{in} & ((f :: [\forall a.a \to a] \to [\forall a.a \to a])\ z)
\end{array}
$$

The safe game:

- Annotations on polymorphic instantiations will always work
- But raises questions about flow of information (look at z's type)

Note: Invariant list constructor

Program example

Check

$$\vdash \forall a.a \to a \leq [\forall a.a \to a] \to [\forall a.a \to a]$$

and infer type $\boxed{[\forall a.a \to a]}$

$$\begin{array}{ll}
\texttt{let} & f\ x = x \\
 & z :: [\forall a.a \to a] = [(f :: (\forall a.a \to a) \to (\forall a.a \to a))\ (\lambda x.x)] \\
\texttt{in} & ((f :: [\forall a.a \to a] \to [\forall a.a \to a])\ z)
\end{array}$$

The safe game:

- Annotations on polymorphic instantiations will always work
- But raises questions about flow of information (look at z's type)

Note: Invariant list constructor

Flow of information in application nodes

Consider again the application rule

$$\frac{\Gamma \vdash t : \boxed{\sigma} \to \rho' \quad \Gamma \vdash u : \sigma}{\Gamma \vdash t\ u : \rho'}\ \text{APP}$$

Information flows from function to argument.

Flow of information in application nodes

Consider again the application rule

$$\frac{\Gamma \vdash t : \boxed{\sigma} \to \rho' \quad \Gamma \vdash u : \sigma}{\Gamma \vdash t\ u : \rho'}\ \text{APP}$$

Information flows from function to argument.
But one could imagine a different rule:

$$\frac{\Gamma \vdash u : \boxed{\sigma} \quad \Gamma \vdash t : \sigma \to \rho'}{\Gamma \vdash t\ u : \rho'}\ \text{APP'}$$

This would type expressions like the *head ids* example of the introduction.

Smart-application

... or even go one step further: Smart-application.

$$\frac{\begin{array}{c} x : \forall \overline{a}\,\overline{b}.\sigma_1 \to \ldots \to \sigma_n \to \sigma_r \in \Gamma \\ \overline{a} \in \mathit{ftv}(\sigma_1, \ldots, \sigma_n) \quad \overline{b} \notin \mathit{ftv}(\sigma_1, \ldots, \sigma_n) \\ \Gamma \vdash u_i : [\overline{a \mapsto \boxed{\sigma_a}}]\sigma_i \quad \vdash [\overline{a \mapsto \sigma_a}, \overline{b \mapsto \boxed{\sigma_b}}]\sigma_r \leq \rho' \end{array}}{\Gamma \vdash x\ u_1 \ldots\ u_n : \rho'}$$

Smart-application

... or even go one step further: Smart-application.

$$\frac{x : \forall \overline{a}\,\overline{b}.\sigma_1 \to \ldots \to \sigma_n \to \sigma_r \in \Gamma \quad \overline{a} \in \mathit{ftv}(\sigma_1, \ldots, \sigma_n) \quad \overline{b} \notin \mathit{ftv}(\sigma_1, \ldots, \sigma_n) \quad \Gamma \vdash u_i : [\overline{a \mapsto \boxed{\sigma_a}}]\sigma_i \quad \vdash [\overline{a \mapsto \sigma_a}, \overline{b \mapsto \boxed{\sigma_b}}]\sigma_r \leq \rho'}{\Gamma \vdash x\ u_1 \ldots\ u_n : \rho'}$$

(But such rules threaten completeness!)

Other heuristics

Modification of subsumption (rule $\textsc{spec}$) and smart-application to two phases:

1. First, the invariant parts of the types are matched and a substitution is created for the quantified variables.
2. Next, that substitution is used for normal subsumption.

For example

$$\not\vdash \forall a.a \to a \leq (\forall a.a \to a) \to Int \to Int$$

With the heuristic however we would:

1. First, create a substitution $[a \mapsto (\forall a.a \to a)]$
2. Next, use the substitution to check that

$$\vdash (\forall a.a \to a) \to (\forall a.a \to a) \leq (\forall a.a \to a) \to Int \to Int$$

Other heuristics

Modification of subsumption (rule SPEC) and smart-application to two phases:

1. First, the invariant parts of the types are matched and a substitution is created for the quantified variables.
2. Next, that substitution is used for normal subsumption.

For example

$$\nvdash \forall a.a \to a \leq (\forall a.a \to a) \to \mathit{Int} \to \mathit{Int}$$

With the heuristic however we would:

1. First, create a substitution $[a \mapsto (\forall a.a \to a)]$
2. Next, use the substitution to check that

$$\vdash (\forall a.a \to a) \to (\forall a.a \to a) \leq (\forall a.a \to a) \to \mathit{Int} \to \mathit{Int}$$

(Heuristics: useful but not robust)

Examples

$$\begin{array}{ll} \texttt{let} & f\ x = x \\ & z :: [\forall a.a \to a] = [f\ (\lambda x.x)] \\ \texttt{in} & (f\ z) \end{array}$$

Let $\vdash \mathit{choose} : \forall a.a \to a \to a$ and $\vdash \mathit{id} : \forall a.a \to a$. Then

$$\begin{array}{lcl} \texttt{let}\ \mathit{auto} & = & \lambda(x :: (\forall a.a \to a)).x\ x \\ \texttt{let}\ f & = & \mathit{choose}\ \mathit{id} \\ \texttt{let}\ f1 & = & \mathit{choose}\ \mathit{id}\ \mathit{succ} \\ \texttt{let}\ f2 & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \\ \texttt{let}\ f3 & = & \mathit{choose}\ (\mathit{id} :: \forall a.(\forall a.a \to a) \to (a \to a))\ \mathit{auto} \\ \texttt{let}\ f4 :: \sigma_{id} \to \sigma_{id} & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \end{array}$$

Examples

$$\begin{array}{ll} \texttt{let} & f\ x = x \\ & z :: [\forall a.a \to a] = [f\ (\lambda x.x)] \\ \texttt{in} & (f\ z) \end{array}$$

Let $\vdash \mathit{choose} : \forall a.a \to a \to a$ and $\vdash \mathit{id} : \forall a.a \to a$. Then

$$\begin{array}{lcl} \texttt{let}\ \mathit{auto} & = & \lambda(x :: (\forall a.a \to a)).x\ x \\ \texttt{let}\ f & = & \mathit{choose}\ \mathit{id} \\ \texttt{let}\ f1 & = & \mathit{choose}\ \mathit{id}\ \mathit{succ} \\ \texttt{let}\ f2 & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \\ \texttt{let}\ f3 & = & \mathit{choose}\ (\mathit{id} :: \forall a.(\forall a.a \to a) \to (a \to a))\ \mathit{auto} \\ \texttt{let}\ f4 :: \sigma_{id} \to \sigma_{id} & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \end{array}$$

$\forall a.(\forall a.a \to a) \to (a \to a)$

Examples

$$\begin{array}{ll} \texttt{let} & f\ x = x \\ & z :: [\forall a.a \to a] = [f\ (\lambda x.x)] \\ \texttt{in} & (f\ z) \end{array}$$

Let $\vdash \mathit{choose} : \forall a.a \to a \to a$ and $\vdash \mathit{id} : \forall a.a \to a$. Then

$$\begin{array}{lcl} \texttt{let}\ \mathit{auto} & = & \lambda(x :: (\forall a.a \to a)).x\ x \\ \texttt{let}\ f & = & \mathit{choose}\ \mathit{id} \\ \texttt{let}\ f1 & = & \mathit{choose}\ \mathit{id}\ \mathit{succ} \\ \texttt{let}\ f2 & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \\ \texttt{let}\ f3 & = & \mathit{choose}\ (\mathit{id} :: \forall a.(\forall a.a \to a) \to (a \to a))\ \mathit{auto} \\ \texttt{let}\ f4 :: \sigma_{id} \to \sigma_{id} & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \end{array}$$

$$\forall a.(a \to a) \to (a \to a)$$

Examples

$$\begin{array}{ll} \texttt{let} & f\ x = x \\ & z :: [\forall a.a \rightarrow a] = [f\ (\lambda x.x)] \\ \texttt{in} & (f\ z) \end{array}$$

Let $\vdash choose : \forall a.a \rightarrow a \rightarrow a$ and $\vdash id : \forall a.a \rightarrow a$. Then

$$\begin{array}{lcl} \texttt{let}\ auto & = & \lambda(x :: (\forall a.a \rightarrow a)).x\ x \\ \texttt{let}\ f & = & choose\ id \\ \texttt{let}\ f1 & = & choose\ id\ succ \\ \texttt{let}\ f2 & = & choose\ id\ auto \\ \texttt{let}\ f3 & = & choose\ (id :: \forall a.(\forall a.a \rightarrow a) \rightarrow (a \rightarrow a))\ auto \\ \texttt{let}\ f4 :: \sigma_{id} \rightarrow \sigma_{id} & = & choose\ id\ auto \end{array}$$

$Int \rightarrow Int$

Examples

$$\begin{array}{ll} \texttt{let} & f\ x = x \\ & z :: [\forall a.a \to a] = [f\ (\lambda x.x)] \\ \texttt{in} & (f\ z) \end{array}$$

Let $\vdash \mathit{choose} : \forall a.a \to a \to a$ and $\vdash \mathit{id} : \forall a.a \to a$. Then

$$\begin{array}{lcl} \texttt{let}\ \mathit{auto} & = & \lambda(x :: (\forall a.a \to a)).x\ x \\ \texttt{let}\ f & = & \mathit{choose}\ \mathit{id} \\ \texttt{let}\ f1 & = & \mathit{choose}\ \mathit{id}\ \mathit{succ} \\ \texttt{let}\ f2 & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \\ \texttt{let}\ f3 & = & \mathit{choose}\ (\mathit{id} :: \forall a.(\forall a.a \to a) \to (a \to a))\ \mathit{auto} \\ \texttt{let}\ f4 :: \sigma_{id} \to \sigma_{id} & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \end{array}$$

fail!

Examples

$$\begin{array}{ll} \texttt{let} & f\ x = x \\ & z :: [\forall a.a \to a] = [f\ (\lambda x.x)] \\ \texttt{in} & (f\ z) \end{array}$$

Let $\vdash \mathit{choose} : \forall a.a \to a \to a$ and $\vdash \mathit{id} : \forall a.a \to a$. Then

$$\begin{array}{lcl} \texttt{let}\ \mathit{auto} & = & \lambda(x :: (\forall a.a \to a)).x\ x \\ \texttt{let}\ f & = & \mathit{choose}\ \mathit{id} \\ \texttt{let}\ f1 & = & \mathit{choose}\ \mathit{id}\ \mathit{succ} \\ \texttt{let}\ f2 & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \\ \texttt{let}\ f3 & = & \mathit{choose}\ (\mathit{id} :: \forall a.(\forall a.a \to a) \to (a \to a))\ \mathit{auto} \\ \texttt{let}\ f4 :: \sigma_{id} \to \sigma_{id} & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \end{array}$$

$\forall a.(\forall a.a \to a) \to (a \to a)$

Examples

$$\begin{array}{ll} \texttt{let} & f\ x = x \\ & z :: [\forall a.a \to a] = [f\ (\lambda x.x)] \\ \texttt{in} & (f\ z) \end{array}$$

Let $\vdash \mathit{choose} : \forall a.a \to a \to a$ and $\vdash \mathit{id} : \forall a.a \to a$. Then

$$\begin{array}{lcl} \texttt{let}\ \mathit{auto} & = & \lambda(x :: (\forall a.a \to a)).x\ x \\ \texttt{let}\ f & = & \mathit{choose}\ \mathit{id} \\ \texttt{let}\ f1 & = & \mathit{choose}\ \mathit{id}\ \mathit{succ} \\ \texttt{let}\ f2 & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \\ \texttt{let}\ f3 & = & \mathit{choose}\ (\mathit{id} :: \forall a.(\forall a.a \to a) \to (a \to a))\ \mathit{auto} \\ \texttt{let}\ f4 :: \sigma_{id} \to \sigma_{id} & = & \mathit{choose}\ \mathit{id}\ \mathit{auto} \end{array}$$

works with last heuristic method!