Type inference in context

Adam Gundry
University of Strathclyde
Microsoft Research PhD Scholar

Conor McBride
University of Strathclyde

James McKinna
Radboud University Nijmegen

MSFP

25 September 2010

Two kinds of problems

- Hindley-Milner type inference
  - λ-calculus with let-definitions
  - Parametric polymorphism

$$\text{let } f \coloneqq \lambda\, x \,.\, x \text{ in } f\ f$$

- First-order unification
  - Solves equations between types

$$\alpha \to \alpha \equiv (\beta \to \beta) \to \gamma$$

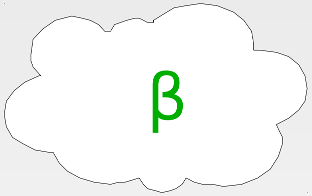
Traditional contexts

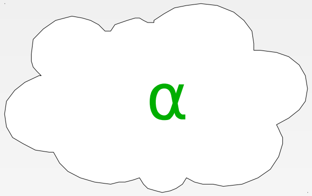
- Contexts explain term variables

$$\Gamma, f : \alpha \to \alpha, t : \beta \to \beta$$

Traditional contexts

- Contexts explain term variables
- Type variables float in space

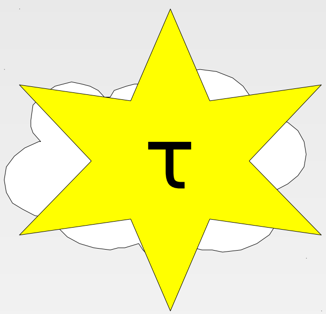


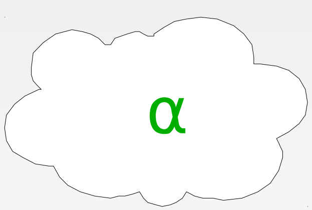


$$\Gamma, f : \alpha \to \alpha, t : \beta \to \beta$$

Traditional contexts

- Contexts explain term variables
- Type variables float in space
- They are given meaning by **substitution**





$$\Gamma, f : \alpha \rightarrow \alpha, t : \tau \rightarrow \tau$$

Type variables in the context

- We add type variables to the context

$$\Gamma, \quad f : \alpha \to \alpha, \quad t : \beta \to \beta$$

Type variables in the context

- We add type variables to the context

$$\Gamma, \alpha := ?, f : \alpha \to \alpha, \beta := ?, t : \beta \to \beta$$

Type variables in the context

- We add type variables to the context
- Give them meaning by **definition**

$$\Gamma, \alpha := ?, f : \alpha \to \alpha, \beta := \tau, t : \beta \to \beta$$

Type variables in the context

- We add type variables to the context
- Give them meaning by **definition**
- Work in the induced equational theory
- Explicitly scoped substitution in triangular form

$$\Gamma, \alpha \coloneqq ?, f : \alpha \to \alpha, \beta \coloneqq \tau, t : \beta \to \beta$$

Declarations and judgments

Declaration	Judgment
$\alpha \coloneqq ?$	$\Gamma \vdash \alpha \equiv \alpha$
$\alpha \coloneqq \tau$	$\Gamma \vdash \alpha \equiv \tau$
$f : \alpha \to \alpha$	$\Gamma \vdash f : \alpha \to \alpha$

Example of unification

To infer the type of the application

$$f\ \ t$$

we have to solve the unification problem

$$\alpha \to \alpha \equiv (\beta \to \beta) \to \gamma$$

where γ is a fresh type variable

Unification algorithm

unify :: Type → Type → StateT Context Maybe ()

Unification algorithm

unify :: Type → Type → Context → Maybe Context

Unification in the context

$$\Gamma, \quad \alpha \coloneqq ?, \quad f, \quad \beta \coloneqq ?, \quad t, \quad \gamma \coloneqq ?$$

$$\alpha \to \alpha \equiv (\beta \to \beta) \to \gamma$$

Unification in the context

$$\Gamma, \quad \alpha := ?, \quad f, \quad \beta := ?, \quad t, \quad \gamma := ?$$

$$\alpha \equiv \beta \to \beta$$

$$\alpha \equiv \gamma$$

Unification in the context

$$\Gamma, \quad \alpha := ?, \quad f, \quad \beta := ?, \quad t, \quad \gamma := ?$$

$\alpha \equiv \beta \to \beta$

$\alpha \equiv \gamma$

Unification in the context

$\Gamma, \quad \alpha := ?, \quad f, \quad \beta := ?, \quad t, \quad \gamma := ?$

$\alpha \equiv \beta \to \beta$

$\alpha \equiv \gamma$

Unification in the context

$$\Gamma, \quad \alpha := ?, \quad f, \quad \beta := ?, \quad t, \quad \gamma := ?$$

$$\alpha \equiv \beta \rightarrow \beta$$

$$\alpha \equiv \gamma$$

Unification in the context

$\Gamma, \quad \alpha := ?, \quad f, \quad t, \quad \gamma := ?$

$\beta := ?$ | $\alpha \equiv \beta \to \beta$

$\alpha \equiv \gamma$

Unification in the context

$$\Gamma, \quad \alpha := ?, \quad f, \quad t, \quad \gamma := ?$$

$\beta := ?$ | $\alpha \equiv \beta \rightarrow \beta$

$\alpha \equiv \gamma$

Unification in the context

$$\Gamma, \quad \beta \coloneqq ?, \quad \alpha \coloneqq ?, \quad f, \quad t, \quad \gamma \coloneqq ?$$

$$\alpha \equiv \beta \to \beta$$

$$\alpha \equiv \gamma$$

Unification in the context

$$\Gamma, \quad \beta := ?, \quad \alpha := \beta \to \beta, \quad f, \quad t, \quad \gamma := ?$$

$$\alpha \equiv \gamma$$

Unification in the context

$$\Gamma, \quad \beta := ?, \quad \alpha := \beta \to \beta, \quad f, \quad t, \quad \gamma := ?$$

$$\alpha \equiv \gamma$$

Unification in the context

$$\Gamma, \quad \beta := ?, \quad \alpha := \beta \to \beta, \quad f, \quad t, \quad \gamma := \alpha$$

Solution strategy

- Refine context in small steps to solve problem
- Easy to verify that each step is sound
- Minimal commitment at each step
- Why does this give most general solutions?

Information increase

- An **information increase** $\theta : \Gamma \sqsubseteq \Delta$ is
  - a substitution from type variables of Γ to types of Δ
  - such that every declaration in Γ
    holds as a judgment in Δ (under the substitution)
- If $\alpha \coloneqq \tau \in \Gamma$ then $\Delta \vdash \theta(\alpha \equiv \tau)$
- We only use the identity substitution,
  but are general with respect to any substitution

Examples of information increase

- Adding fresh variables

$$\Gamma \sqsubseteq \Gamma, \alpha \coloneqq ?, \beta \coloneqq ?$$

- Defining a previously undefined type variable

$$\Gamma, \alpha \coloneqq ? \sqsubseteq \Gamma, \alpha \coloneqq \tau$$

- Substituting out a definition

$$[\tau/\alpha] : \Gamma, \alpha \coloneqq \tau, \beta \coloneqq \alpha \sqsubseteq \Gamma, \beta \coloneqq \tau$$

Stability

- A judgment is **stable** if it is preserved by information increase
- “Once solved, always solved”
- Stability by construction: context access just looks up facts about variables
- Minimal-commitment solutions to stable problems are most general

Type inference

infer :: Term → StateT Context Maybe Type

Type inference

infer :: Term → Context → Maybe (Type, Context)

Type inference: term variables

$$x :: \forall\, \alpha\, \beta\, .\, \alpha \to \beta \to \alpha \quad \in \quad \Gamma$$

- Contexts give type-schemes to term variables

Type inference: term variables

$$\Gamma \qquad\qquad \vdash \; x :: \forall\, \alpha\, \beta\, .\, \alpha \to \beta \to \alpha$$

- Instantiate type-scheme with fresh variables

Type inference: term variables

$$\Gamma, \alpha' \coloneqq ? \qquad \vdash \; x :: \forall \beta \, . \, \alpha' \to \beta \to \alpha'$$

- Instantiate type-scheme with fresh variables

Type inference: term variables

$$\Gamma, \alpha' := ?, \beta' := ? \vdash x : \alpha' \to \beta' \to \alpha'$$

- Instantiate type-scheme with fresh variables

Generalisation

- How to generalise types in let-expressions?
- Traditionally, compare sets of free variables

$$\frac{A \vdash e' : \tau' \qquad A_x \cup \{x : \sigma\} \vdash e : \tau}{A \vdash \text{let } x := e' \text{ in } e : \tau} \; \sigma = gen(A, \tau')$$

$$gen(A, \tau) = \begin{cases} \forall \vec{\alpha_i}.\tau & (FV(\tau) \setminus FV(A) = \{\alpha_1, \cdots, \alpha_n\}) \\ \tau & (FV(\tau) \setminus FV(A) = \emptyset) \end{cases}$$

Generalisation

- Structure on type variables makes it easy
- 'Skim off' type variables from the local, unconstrained end of the context
- Use a marker to record where to stop
- Unification may move variables past the marker

Type inference: let expressions

- To infer the type of let expressions:

$$\text{let } y \coloneqq t \text{ in } e$$

- Place a marker in the context
- Infer the type of the definition t
- Generalise over type variables
- Extend the context with a type-scheme for y
- Infer the type of the body e

Type inference: let expressions

$\Gamma \vdash \text{let } f \coloneqq \lambda x . x \text{ in } f\ f : ?$

Type inference: let expressions

$$\Gamma \,; \qquad \vdash \text{let } f \coloneqq \lambda x\,.\,x \text{ in } f\ f : ?$$

- Place a marker ; in the context
- This records where to stop generalising

Type inference: let expressions

$$\Gamma\,; \qquad\qquad \vdash \lambda x . x : ?$$

- Infer the type of the let-definition

Type inference: let expressions

$$\Gamma \; ; \; \alpha \coloneqq \; ? \qquad \vdash \; \lambda \, x \, . \, x : \alpha \to \alpha$$

- Infer the type of the let-definition

Type inference: let expressions

$$\Gamma\ ;\ \alpha \coloneqq ? \qquad \vdash\ \lambda\, x\, .\, x : \alpha \to \alpha$$

- Generalise back to the marker

Type inference: let expressions

$\Gamma \; ; \qquad \vdash \lambda x . x :: \forall \alpha . \alpha \to \alpha$

- Generalise back to the marker

Type inference: let expressions

$$\Gamma, f :: \forall \alpha . \alpha \to \alpha \vdash f\ f : ?$$

- Assign type-scheme to the let-bound variable
- Infer the type of the let-body

Giving up some freedom

- Typing for let expressions is non-compositional
- We must restrict the information increase relation so terms have principal types
- Forbid assignment of more general types to variables in the context

$$\Gamma, x : \alpha \not\sqsubseteq \Gamma, x :: \forall \alpha . \alpha$$

What have we achieved?

- A methodology for problem solving in contexts
- Connected soundness and generality of algorithms for problem solving
- More intuitive account of generalisation

How to solve problems in contexts

- Define properties of variables
- Define judgments: stable by construction
- Show solution steps are information increases
- Restrict freedom so unique solutions exist
- Minimal commitment yields principal solutions

Future directions

- Formal correctness proof?
- Investigate more complex type systems
- Undecidable constraint systems
- Represent syntactic context explicitly (zipper)

References

- D. Clément et. al.
  A simple applicative language: mini-ML.
  LFP '86.
- L. Damas and R. Milner.
  Principal type-schemes for functional programs.
  POPL '82.
- J. Dunfield. Greedy bidirectional polymorphism.
  ML '09.

References

- W. Naraschewski and T. Nipkow.
  Type inference verified: Algorithm W in Isabelle/HOL.
  J. Automated Reasoning, 23(3):299-318, 1999.
- M. Wand. A Simple Algorithm and Proof for Type Inference.
  Fundamenta Informaticae 10:115-122, 1987.
- J. B. Wells. The essence of principal typings.
  ICALP '02.

Free monad on derivations

- Stability extends substitution on types to substitution on typing derivations
- Derivations have a free monad structure:
  - Return takes a variable to the derivation that looks up the variable in the context
  - Bind is substitution of sub-derivations