Stratified Type Inference for Generalized Algebraic Data Types

François Pottier and Yann Régis-Gianas

POPL 2006, Charleston
January 13, 2006
What are GADTs?

- With GADTs, data constructors can have more precise type schemes than with usual algebraic data types.
- Here is a GADT declaration for the abstract syntax of a simply typed toy language:

```
type term :: ⋆ → ⋆ =
  | Lit : int → term int
  | Inc : term int → term int
  | Pair : ∀γγ'.term γ × term γ' → term (γ × γ')
  | Fst : ∀γγ'.term (γ × γ') → term γ
  | Snd : ∀γγ'.term (γ × γ') → term γ'
```
GADTs give rise to local equations

This is a tagless interpreter for terms:

\[
\text{fix } (\text{eval} : \forall \alpha. \text{term } \alpha \rightarrow \alpha). \lambda t. \\
\text{case } t \text{ of } \\
| \text{Inc } t' \rightarrow (\ast \text{Inc : term } \text{int} \rightarrow \text{term } \text{int} \text{ hence } \alpha = \text{int } \ast) \\
\text{eval } t' + 1 \\
| \ldots \\
\]
The problem of type inference

Roughly speaking, what makes type inference for GADTs difficult is to answer these questions:

- What local equations are available?
- Where and how are they used?
The problem of type inference

- What local equations are available?

\[
\text{type eq} :: \star \to \star \to \star =
\]
\[
Eq : \forall \alpha.\text{eq } \alpha \alpha
\]

let \( \text{cast } t \ x = \text{case } t \text{ of } Eq \to x \)

- Assuming \( t : \text{eq } \tau_1 \tau_2 \), the local equation is \( \tau_1 = \tau_2 \). Yet, during type inference, \( \tau_1 \) and \( \tau_2 \) are unknown, and so is the equation ...
The problem of type inference

► Where and how are they used?

\[
\text{type } eq :: \star \to \star \to \star = \\
Eq : \forall \alpha. eq \, \alpha \, \alpha
\]

\[
\text{let } \text{cast } t \, x = \text{case } t \text{ of } Eq \to x
\]

► Assuming \( t : eq \, \tau_1 \, \tau_2 \) and \( x : \tau_1 \), then the local equation is \( \tau_1 = \tau_2 \) and the return type can be:

► \( \tau_2 \) by using the local equation.
► \( \tau_1 \) by not using the local equation.

► Finding out about all choices requires knowing which local equations are available.
The problem of type inference

- The existence of choices means that there is no principal type:

  \[
  \text{type } eq :: \star \to \star \to \star = \\
  Eq : \forall \alpha. eq \alpha \alpha
  \]

  let cast \ t \ x = case \ t \ of \ Eq \ \to \ x

- These two ML type schemes are valid for cast and are incomparable:
  - \ \forall \gamma'\gamma''. eq \gamma' \gamma'' \to \gamma \to \gamma
  - \ \forall \gamma' eq \gamma' \to \gamma \to \gamma'

- Thus, user annotations seem unavoidable.
Is life simpler in everyday programs?

With just a few type annotations, type inference becomes very much like type checking:

\[
\text{fix } (eval : \forall \alpha. \text{term } \alpha \rightarrow \alpha). \lambda t. \\
\text{case } t \text{ of } \\
| \text{Inc } t \rightarrow (\ast \alpha = \text{int } \ast) \\
| \text{eval } t + 1 \\
| \ldots 
\]
Stratified type inference

Two natural approaches:

▶ Implement an incomplete type inference engine dealing with ML and GADTs simultaneously (Sulzmann et al)

▶ or (the approach we will describe today):

1. Design a restricted core language where enough annotations are required to make complete type inference as easy as in ML;
2. Design an incomplete but predictable algorithm to propagate user annotations.

This approach is followed by Peyton Jones et al, except they do not clearly separate 1 and 2.
Stratified type inference at a glance

1. SOURCE
2. Preprocessor to propagate annotations
3. MLGX
4. Type Inference
   - Yes
   - No
Stratified type inference at a glance
MLGX: ML with explicit typing of GADTs

First restriction:

Case constructs must be annotated.

\[
\text{case } (t : \tau) \text{ of } \ldots
\]

- Only this annotation will be used to deduce the local equations, not the inferred type. Thus, local equations are known everywhere even before type inference begins.
Second restriction:

An explicit coercion is required to use an equation.

\((t : \tau_1 \triangleright \tau_2)\)

- This coercion is valid only if \(\tau_1 = \tau_2\) is implied by the local equations.
- This implication test is possible because the local equations are known everywhere.
- Once this test is done, this coercion can be viewed as an application of the identity at type \(\tau_1 \rightarrow \tau_2\) to \(t\).
Properties of MLGX

- In MLGX, type inference is simple:
  1. Every coercion is checked.
  2. Viewing coercions as constants, standard ML type inference is performed.

- MLGX has principal types.

- Every ML program is well-typed in MLGX.
Our example in MLGX

- Our example must be annotated to be well-typed in MLGX:

\[
\text{fix } (eval : \forall \alpha. \text{term } \alpha \rightarrow \alpha). \lambda t. \\
\quad \text{case } (t : \text{term } \alpha) \text{ of} \\
\quad \quad \mid \text{Inc } t \rightarrow (eval \ t + 1 : (\text{int } \triangleright \alpha)) \\
\quad \quad \mid \ldots
\]
Our example must be annotated to be well-typed in MLGX:

\[
\begin{align*}
\text{fix } (\text{eval} : \forall \alpha.\text{term } \alpha \rightarrow \alpha).\lambda t. \\
\quad \text{case } (t : \text{term } \alpha) \text{ of} \\
\quad \quad | \text{Inc } t \rightarrow (\text{eval } t + 1 : (\text{int } \triangleright \alpha)) \\
\quad \quad | \ldots
\end{align*}
\]

The red annotations are hard to infer whereas it seems easy to deduce the blue ones from the red ones.

⇒ Why not do so automatically?
Stratified type inference at a glance
Local shape inference as a preprocessor

- The idea is to design a preprocessor to deduce the blue annotations from the red ones.

- This preprocessor will necessarily be incomplete, hence it should be predictable.

- How to achieve predictability?
  - By preventing long-distance unification.
  - We introduce an algebra of shapes to support local propagation of information.
Shapes

- A shape looks exactly like a type scheme, but is interpreted as a type approximation.

- The shape $\gamma . \gamma \rightarrow \gamma$ describes both the identity function and the integer successor function.

- Shapes are closed. An unknown is never shared between two shapes. This prevents long-distance unification.

- Shapes can be combined by first order unification:

  $$(\gamma . \gamma \rightarrow \gamma) \sqcup (\gamma' . \gamma' \rightarrow \text{int}) = \text{int} \rightarrow \text{int}$$
Shape inference in action

- Shape inference is rigorously defined in the paper.
- Shape inference is able to deduce the type of $t$, to deduce the type of the case construct, and to insert the necessary annotation and coercion:

  \[
  \text{fix} \ (eval : \forall \alpha. \text{term } \alpha \to \alpha). \lambda t. \\
  \text{case} \ (t : \text{term } \alpha) \text{ of} \\
  \quad \mid \text{Inc } t \to (eval t + 1 : (\text{int} \Downarrow \alpha)) \\
  \quad \mid \ldots
  \]
Thanks to shape unification, local shape inference is robust in the face of minor changes:

\[
\text{fix } (\text{eval} : \forall \alpha. \text{term} \alpha \to \alpha). \lambda t. \\
\text{case } (\text{id } t : \text{term } \alpha) \text{ of } \\
| \text{inc } t \to (\text{eval } t + 1 : (\text{int } \triangleright \alpha)) \\
| \ldots
\]

Indeed, \((\gamma, \gamma \to \gamma) \sqcup (\gamma', \text{term } \alpha \to \gamma')\) is \text{term } \alpha \to \text{term } \alpha.
Summary about MLGX

- MLGX is a **simple** extension of ML that introduces GADTs while preserving principal types.
- It is **robust** in the sense that there are not many ways of altering its design.
Summary about local inference

- Stratified type inference enables a modular extension of ML type inference. Because local shape inference is incomplete, it is natural to keep it apart from MLGX.

- We have experimented with two different shape inference algorithms based on the same shape toolbox. One is inspired from GHC’s wobbly type inference. The other is strictly more precise.

- Both algorithms are sound in a sense defined in the paper.

- This is an original use of local inference in a language à la ML (as opposed to System F).