Compositional Explanation of Types and Algorithmic Debugging of Type Errors

Olaf Chitil
The University of York, UK
Example

reverse [] = []
reverse (x:xs) = reverse xs ++ x

last xs = head (reverse xs)
init = reverse . tail . reverse

rotateR xs = last xs : init xs

Hugs 98:

Error : Type error in application
*** Expression : last xs : init xs
*** Term : last xs
*** Type : [a]
*** Does not match: a
*** Because : unification would give infinite type
Problems:

- wrong error location
- where do the two types come from?
- scope of type variables?
- meaning of error message?
Milner's algorithm W

- introduces globally scoped type variables
- globally updates variables
- has left-right information flow
  \[ f(\text{not } x)(x++ \ "demo") \]
Hindley-Milner Type Inference Tree

\[
\begin{align*}
\{x :: \text{Bool}\} \vdash x :: \text{Bool} & \quad \{x\} \vdash \neg x :: \text{Bool} \\
\{x :: \text{Bool}\} \vdash x :: \text{Bool} & \quad \{x :: \text{Bool}\} \vdash \neg x :: \text{Bool} \\
\{x :: \text{Bool}\} \vdash (x, \neg x) :: (\text{Bool, Bool})
\end{align*}
\]

- not compositional because of environment
- no proof that there exists no more general type
Solution: Principal Typeings

**principal type:** type for given expression + type environment

\{ x :: \text{Bool} \} \vdash x :: \text{Bool}

**principal typing:** type environment + type for given expression typing

\{ x :: \alpha \} \vdash x :: \alpha
An Inference Tree of Principal Typings is Compositional

\[
\begin{align*}
\text{Ex} : & \text{Bool} \\
\text{Ex} : & \text{Bool} \\
\text{Ex} : & (x, \text{not } x) : (\text{Bool}, \text{Bool})
\end{align*}
\]
But could be a let-bound, polymorphic variable

\[ \forall a, d: 3 \vdash (x, \text{not } x) : (a, \text{Bool}) \]

the Hindley-Milner system doesn't have principal typings [Wiedl '96]

Solution: separate environment for let-bound variables [Mitchell '96]
The Explanation Graph

\[
\begin{align*}
\{x :: \alpha\} & \vdash x :: \alpha \\
\{ \{\} \vdash x :: \alpha\} & \vdash \text{let } x = x \\
\{\} & \vdash x :: \alpha \\
\{\} & \vdash \text{not :: Bool} \\
\{\} & \vdash \text{not x :: Bool} \\
\{\} & \vdash (x, \text{not x}) :: (\alpha, \text{Bool})
\end{align*}
\]

not completely syntax-directed, but compositional
Navigation through the graph

Type error in (last xs) : (init xs)

because

Expressions:

 Cáj -> [a]

Types:

[La] with xs
Navigation at the Level of Polymorphic Variables

Type error in \( (\text{last } xs) : (\text{init } xs) \)

because

\[
\text{last} :: [\text{[a]}] \rightarrow \text{a} \\
\text{init} :: [\text{[a]}] \rightarrow [\text{a}]
\]

> 1
last :: [[a]] → a

because

reverse :: [[a]] → [a]

because

> 1

> switch to level of all program fragments
reverse :: [a] -> [a]

because

Equation: reverse [] = [] reverse (x:xs) = ...

with reverse [b] -> [c]

[[a]] -> [a]

> 2

Equation: reverse (x:xs) = ...
with reverse [[a]] -> [a]

because

Lhs/Rhs: reverse (x:xs)

Types: b

with reverse [c] -> b

x c

xs [c]

> 2
Expression : \((\text{reverse } \text{xs}) \, \text{++} \, \text{x}\)

Type : \([a] \)

with reverse \(d \rightarrow [a] \)

\(\times \quad [a] \)

\(\times \quad d \)

because

Expressions : \((\text{++}) \, (\text{reverse } \text{xs}) \, \text{x}\)

Types : \([b] \rightarrow [b] \quad c \)

with reverse \(a \rightarrow [b] \)

\(\times \quad c \)

\(\timess \quad a \)
Algorithmic Debugging

Shapiro '83

\[ \text{Equations and symbols with checks and crosses} \]
Type error in: \((\text{last } x)\) : \((\text{init } x)\)

\(\text{last} :: [\text{a}] \to \text{a}\)
Is intended type an instance? (y/n) n

\(\text{reverse} :: [\text{a}] \to [\text{a}]\)
Is intended type an instance? (y/n) n

Error located! In definition of \(\text{reverse}\)
Switch to level of all program fragments

\(\text{reverse} :: [\text{b}] \to [\text{c}]\)
Is intended type an instance? (y/n) Y

\(\text{reverse} :: [\text{a}] \to [\text{a}]\)
Is intended type an instance? (y/n) n
\[
\text{reverse } (x:xs) \quad :: \quad b
\]
\[
\text{reverse} \quad :: \quad [c \rightarrow b]
\]
\[
\times \quad :: \quad c
\]
\[
\times s \quad :: \quad [c]
\]

Are intended types an instance? (y/n) \quad Y

\[
(\text{reverse } \times s) \; \; \times \quad :: \quad [a]
\]
\[
\text{reverse} \quad :: \quad d \rightarrow [a]
\]
\[
\times \quad :: \quad [a]
\]
\[
\times s \quad :: \quad d
\]

Are intended types an instance? (y/n) \quad n
(++) (reverse xs) :: [b] → [b]
reverse        :: a → [b]
xs             :: a

Are intended types an instance? (y/n)  y

Error located! Wrong expression:
(reverse xs) ++ x
Implementation

prototype based on Mark Jones’ type checker for core Haskell

efficiency can be improved by

• piecemeal generation of explanation graph

• combination with algorithm W
Summary

- Compositionality is the key to good explanations
- Principal typings instead of principal types
- Graph for free navigation and algorithmic debugging

Future Work

- Haskell class system
- Graphical user interface