A Generalization of Hybrid Let-Polymorphic Type Inference
Algorithms

Oukseh Lee
(Co-work with Kwangkeun Yi)

Research On Program Analysis System
http://ropas.kaist.ac.kr
Korea Advanced Institute of Science and Technology

December 18, 2000
The *de facto* standard type inference algorithm of the Hindley/Milner type system [Mil78,DM82].

- Bottom-up or context-insensitive.
- Checks conflicts after inferring types of sub-expressions independently.

Problem: sometimes helpless type error messages.

```
# fun fac n = if n=0 then 1 else n * fac(n=1);;
This expression has type int -> int,
but is used with type bool -> int.
```

Suggested solutions: alternative algorithms.
• Alternative type inference algorithm.
  – Top-down or context-sensitive.
  – Carries a type constraint down to its sub-or-sibling expressions.

• Different type error messages.

```ocaml
# fun fac n = if n=0 then 1 else n * fac(n=1);;
its type is 'a -> 'b -> bool,
but its type is expected as 'a -> 'b -> int.
```
Other Hybrid Algorithms

- Objective Caml 2.04 employs a variant of $\mathcal{M}$.
  
  ```
  # fun fac n = if n=0 then 1 else n * fac(n=1);;
  its type is bool,
  but its type is expected as int.
  ```

- Standard ML of New Jersey employs a variant of $\mathcal{W}$.
  
  ```
  # fun fac n = if n=0 then 1 else n * fac(n=1);;
  its operator domain is int type,
  but its operand is bool type.
  ```

- They are not formally investigated.

- Other hybrid algorithms are possible.
A Generalization Is Necessary

- To formally investigate existing type inference algorithms.
  - Whether they are sound and complete.
  - What are the difference from other algorithms.
- To avoid efforts to design new sound and complete type inference algorithms.
What is the Difference?

\[
\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 \ e_2 : \tau_2}
\]

\[
\begin{align*}
\rho \\
\downarrow \\
e_1 \ e_2 \\
\downarrow \\
e_1 & \quad e_2 \\
\beta \rightarrow \rho & \quad \beta & \quad \beta
\end{align*}
\]

\[M\]

\[
\begin{align*}
\rho \\
\downarrow \\
e_1 \ e_2 \\
\downarrow \\
e_1 & \quad e_2 \\
\tau_1 & \quad \beta \\
\downarrow & \quad \downarrow
\end{align*}
\]

OCaml’s

\[
\begin{align*}
\rho \\
\downarrow \\
e_1 \ e_2 \\
\downarrow \\
e_1 & \quad e_2 \\
\tau_1 & \quad \tau_2 \\
\downarrow & \quad \downarrow
\end{align*}
\]

\[W\]
What is the Difference?

\[ \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1 \]
\[ \Gamma \vdash e_1 \ e_2 : \tau_2 \]

- Difference: information amount of type constraint and place of unification
\[ M(\Gamma, e_1 e_2, \rho) = \]
\[
\text{let } S_1 = M(\Gamma, e_1, \beta \to \rho) \\
S_2 = M(S_1 \Gamma, e_2, S_1\beta) \\
in \ S_2S_1
\]
\[ M(\Gamma, e_1, e_2, \rho) = \]
\[
\text{let } S_1 = M(\Gamma, e_1, \beta \rightarrow \rho) \\
S_2 = M(S_1 \Gamma, e_2, S_1 / \beta) \\
\text{in } S_2 S_1
\]

- How to control the information amount of type constraints?
  - Loosen the type constraints.

- How to control the places of unifications?
  - Insert an unification into every possible point.
  - Check the result types against the loosened types.

- Every loosened type constraint has to be compensated.
\[ M(\Gamma, e_1 e_2, \rho) = \]
\[ \text{let } S_1 = M(\Gamma, e_1, \beta \rightarrow \rho) \]
\[ S_2 = M(S_1\Gamma, e_2, S_1\beta) \]
\[ \text{in } S_2 S_1 \]

\[ G(\Gamma, e_1 e_2, \rho) = \]
\[ \text{let } S_1 = G(\Gamma, e_1, \theta_1) \quad (1) \theta_1 \geq \beta \rightarrow \rho \]
\[ S_2 = U(S_1\theta_1, \theta_2) \quad (2) \theta_2 \geq S_1(\beta \rightarrow \rho) \]
\[ S_3 = G(S_2S_1\Gamma, e_2, \theta_3) \quad (3) \theta_3 \geq S_2S_1\beta \]
\[ S_4 = U(S_3S_2S_1\theta_1, S_3S_2S_1(\beta \rightarrow \rho)) \]
\[ S_5 = U(S_4S_3\theta_3, S_4S_3S_2S_1\beta) \]
\[ \text{in } S_5S_4S_3S_2S_1 \]
\[ M: \text{Revisited} \]

\[ G(\Gamma, e_1 e_2, \rho) = \]

let \( S_1 = G(\Gamma, e_1, \beta \rightarrow \rho) \)

\( S_2 = U(S_1(\beta \rightarrow \rho), S_1(\beta \rightarrow \rho)) \)

\( S_3 = G(S_2S_1\Gamma, e_2, S_2S_1\beta) \)

\( S_4 = U(S_3S_2S_1(\beta \rightarrow \rho), S_3S_2S_1(\beta \rightarrow \rho)) \)

\( S_5 = U(S_4S_3S_2S_1\beta, S_4S_3S_2S_1\beta) \)

\( \text{in } S_5S_4S_3S_2S_1 \)

(1) \( \theta_1 = \beta \rightarrow \rho \geq \beta \rightarrow \rho \)

(2) \( \theta_2 = S_1(\beta \rightarrow \rho) \geq S_1(\beta \rightarrow \rho) \)

(3) \( \theta_3 = S_2S_1\beta \geq S_2S_1\beta \)
\[ \mathcal{G}(\Gamma, e_1, e_2, \rho) = \]

let 
\[ S_1 = \mathcal{G}(\Gamma, e_1, \beta_1) \]
\[ S_2 = \mathcal{U}(S_1\beta_1, \beta_2) \]
\[ S_3 = \mathcal{G}(S_2S_1\Gamma, e_2, \beta_3) \]
\[ S_4 = \mathcal{U}(S_3S_2S_1\beta_1, S_3S_2S_1(\beta \rightarrow \rho)) \]
\[ S_5 = \mathcal{U}(S_4S_3\beta_3, S_4S_3S_2S_1\beta) \]

in 
\[ S_5S_4S_3S_2S_1 \]

\[ \theta_1 = \beta_1 \geq \beta \rightarrow \rho \]
\[ \theta_2 = \beta_2 \geq S_1(\beta \rightarrow \rho) \]
\[ \theta_3 = \beta_3 \geq S_2S_1\beta \]
\[ \mathcal{G}(\Gamma, e_1 e_2, \rho) = \]

let \( S_1 = \mathcal{G}(\Gamma, e_1, \beta_1) \)

\( S_2 = \mathcal{U}(S_1 \beta_1, S_1(\beta \rightarrow \rho)) \)

\( S_3 = \mathcal{G}(S_2S_1 \Gamma, e_2, S_2S_1 \beta) \)

\( S_4 = \mathcal{U}(S_3S_2S_1 \beta_1, S_3S_2S_1(\beta \rightarrow \rho)) \)

\( S_5 = \mathcal{U}(S_4S_3S_2S_1 \beta, S_4S_3S_2S_1 \beta) \)

in \( S_5S_4S_3S_2S_1 \)
Another Algorithm Is Possible: \( H \)

\[ \mathcal{G}(\Gamma, e_1, e_2, \rho) = \]

let \( S_1 = \mathcal{G}(\Gamma, e_1, \beta \rightarrow \beta_1) \)  
\[ (1) \quad \theta_1 = \beta \rightarrow \beta_1 \geq \beta \rightarrow \rho \]

\( S_2 = \mathcal{U}(S_1(\beta \rightarrow \beta_1), S_1(\beta \rightarrow \rho)) \)  
\[ (2) \quad \theta_2 = S_1(\beta \rightarrow \rho) \geq S_1(\beta \rightarrow \rho) \]

\( S_3 = \mathcal{G}(S_2S_1\Gamma, e_2, S_2S_1\beta) \)  
\[ (3) \quad \theta_3 = S_2S_1\beta \geq S_2S_1\beta \]

\( S_4 = \mathcal{U}(S_3S_2S_1(\beta \rightarrow \beta_1), S_3S_2S_1(\beta \rightarrow \rho)) \)

\( S_5 = \mathcal{U}(S_4S_3S_2S_1\beta, S_4S_3S_2S_1\beta) \)

in \( S_5S_4S_3S_2S_1 \)
Fact 1. \( \mathcal{W}, \mathcal{M}, \text{OCaml 2.04's}, \text{SML/NJ's}, \text{and } \mathcal{H} \) are instances of \( G \).
Every Instance Is Sound and Complete

Theorem 1 (Soundness)

\[ S = G(\Gamma, e, \rho) \implies S\Gamma \vdash e : S\rho. \]

Theorem 2 (Completeness)

\[ S\Gamma \vdash e : S\rho \implies S = G(\Gamma, e, \rho). \]
Relative Earliness \textsuperscript{[LY98]}

A stops earlier than $A'$ $\Leftrightarrow$ A’s call string is shorter than $A'$’s

$(\|A(\Gamma, e, \rho)\| \leq \|A'(\Gamma, e, \rho)\|)$. 

![Diagram showing relative earliness between two call strings with A and A']
Definition 1 (More Restraining Instance)

\[ A \subseteq A' \]

\[ \updownarrow \]

For each corresponding pair of

\[ \theta_i \geq \rho_i \] during \( A(\Gamma, e, \rho) \) and

\[ \theta'_i \geq \rho'_i \] during \( A'(\Gamma, e, \rho) \),

if \( \rho_i \leq \rho'_i \), then \( \theta_i \leq \theta'_i \)

Lemma 1

\[ \mathcal{M} \subseteq \mathcal{H} \subseteq \text{OCaml's} \subseteq \text{SML/NJ's} \subseteq \mathcal{W} \]
Theorem 3.

\[ A \subseteq A' \implies |[A(\Gamma, e, \rho)]| \leq |[A'(\Gamma, e, \rho)]| \]

Thus,

Corollary 1.

\[
|[\mathcal{M}(\Gamma, e, \rho)]| \leq |[\mathcal{H}(\Gamma, e, \rho)]| \\
\leq |[\text{OCaml's}(\Gamma, e, \rho)]| \\
\leq |[\text{SML/NJ's}(\Gamma, e, \rho)]| \\
\leq |[\mathcal{W}(\Gamma, e, \rho)]|
\]
Summary

- A generalized let-polymorphic type inference algorithm: $G$
  - Every instance is sound and complete.
  - More restraining instance stops earlier.
  - $\mathcal{W}$, $\mathcal{M}$, $\mathcal{H}$, SML/NJ’s, and OCaml’s are its instances.
  - Easy to generate new instances.

- Further generalization is possible by loosening $\Gamma$ in $G(\Gamma, e, \rho)$.

- Sulzmann [Sul00] suggested a general framework in constraint forms.