A Generalized Let-Polymorphic Type Inference Algorithm

Oukseh Lee and Kwangkeun Yi
{cookcu; kwang}@ropas.kaist.ac.kr

Abstract

We present a generalized let-polymorphic type inference algorithm, prove that any of its instances is sound and complete with respect to the Hindley/Milner let-polymorphic type system, and find a condition on two instance algorithms so that one algorithm should find type errors earlier than the other.

By instantiating the generalized algorithm with different parameters, we can achieve not only the two opposite algorithms (the bottom-up standard Algorithm $W$ and the top-down folklore algorithm $M$) but also other various hybrid algorithms that avoid their extremities in type-checking ($W$ fails too late, while $M$ fails too early). Such hybrid algorithms’ soundness, completeness, and their relative earliness in detecting type-errors follow automatically. The set of hybrid algorithms that come from the generalized algorithm is a superset of those used in the two most popular ML compilers, SML/NJ and OCaml.

1 Introduction

The Hindley/Milner let-polymorphic type system[Mil78]'s two opposite algorithms (the standard bottom-up Algorithm $W$ [DM82, Mil78] and the folklore top-down algorithm $M$ [LY98]) are two extremes in type-checking. Algorithm $W$ is context-insensitive, finding type errors too late, while algorithm $M$ is as much context-sensitive as possible, finding type errors too early. $W$ fails only at an application expression where its two sub-expressions (function and argument) have conflicting types. Because of this, an erroneous expression is often successfully type-checked (context-insensitively) long before its consequence collides at an application expression. On the other hand, $M$ carries the most informative type-constraint (or expected type) implied by the context of an expression down to its sub-or-sibling expressions. It fails when the current expression’s type cannot satisfy the carried type constraint. As an example to show the difference between the two algorithms, consider an application expression “1 $2$. $W$ fails at the top expression after having successfully type-checked the two sub-expressions, while $M$ fails at the left expression 1 because its type int conflicts with a function type expected from the context (an application).

In realistic compiler systems we need algorithms that avoid this extreme behaviors, but there exists no systematic way to design such hybrid algorithms. Already, some combinations of the two algorithms have been used in the SML/NJ[sml99] and OCaml[LRVD99] compilers, yet without proofs on their soundness and completeness and on how they differ on their type-checking behaviors. In order to systematically explore other hybrid algorithms, as well as to justify the existing hybrid algorithms, we need a framework (1) for integrating the two opposite algorithms into one algorithm that avoids their extremities in type-checking, (2) for assuring that such an integrated algorithm is still sound and complete, and (3) for measuring, if possible, relatively how early (or late) it find type errors.

This work is supported by Creative Research Initiatives of the Korean Ministry of Science and Technology.
We present a generalized let-polymorphic type inference algorithm, prove that any of its instances is sound and complete with respect to the Hindley/Milner let-polymorphic type system, and find a condition on two instance algorithms so that one algorithm should find type errors earlier than the other. By instantiating the generalized algorithm with different parameters, we can achieve not only the two opposite algorithms (W and M) but also other various hybrid algorithms that avoid their extremities in type-checking. Such hybrid algorithms' soundness and completeness follow automatically, and their relative earliness in detecting type-errors can be decided by checking a simple condition. The set of hybrid algorithms that come from the generalized algorithm is a superset of the existing hybrid algorithms in SML/NJ [smi99] and OCaml [LRVD99].

1.1 Notation

We use the same conventional notation as used in [LY98]. Vector \( \vec{\alpha} \) is a shorthand for \( \{ \alpha_1, \cdots, \alpha_n \} \), and \( \forall \vec{\alpha}. \tau \) is for \( \forall \alpha_1 \cdots \alpha_n. \tau \). Equality of type schemes is up to renaming of bound variables. For a type scheme \( \sigma = \forall \vec{\alpha}. \tau \), the set \( ftv(\sigma) \) of free type variables in \( \sigma \) is \( ftv(\tau) \setminus \vec{\alpha} \), where \( ftv(\tau) \) is the set of type variables in type \( \tau \). For a type environment \( \Gamma \), \( ftv(\Gamma) = \bigcup_{x \in dom(\Gamma)} ftv(\Gamma(x)) \). A substitution \( \{ \tau_i/\alpha_i \mid 1 \leq i \leq n \} \) substitutes type \( \tau_i \) for type variable \( \alpha_i \). We write \( \{ \vec{\tau}/\vec{\alpha} \} \) as a shorthand for a substitution \( \{ \tau_i/\alpha_i \mid 1 \leq i \leq n \} \), where \( \vec{\alpha} \) and \( \vec{\tau} \) have the same length \( n \) and \( \vec{R} \) for \( \{ \vec{R}_{\alpha_1}, \cdots, \vec{R}_{\alpha_n} \} \). For a substitution \( S \), the support \( supp(S) \) is \( \{ \alpha \mid S\alpha \neq \alpha \} \), and the set \( itv(S) \) of involved type variables is \( \{ \alpha \mid R\alpha \supseteq supp(S) \wedge \alpha \in \{ \beta \} \cup ftv(S(\beta)) \} \). For a substitution \( S \) and a type \( \tau \), \( S\tau \) is the type resulting from applying every substitution component \( \tau_i/\alpha_i \) in \( S \) to \( \tau \). Hence, \( \{ \} \tau = \tau \). For a substitution \( S \) and a type scheme \( \sigma \), \( S\sigma = \forall \vec{\beta}. S(\vec{\beta}/\vec{\alpha})\tau \), where \( \vec{\beta} \cap \{ itv(S) \cup ftv(\sigma) \} = \emptyset \). For a substitution \( S \) and a type environment \( \Gamma \), \( SF = \{ x \mapsto S\sigma \mid x \mapsto \sigma \in \Gamma \} \). The composition of substitutions \( S \) followed by \( R \) is written as \( RS \), which is \( \{ \vec{R}(S\alpha) \mid \alpha \in supp(S) \} \cup \{ R\alpha/\alpha \mid \alpha \in supp(R) \setminus supp(S) \} \). Two substitutions \( S \) and \( R \) are equal if and only if \( S\alpha = R\alpha \) for every \( \alpha \in supp(S) \cup supp(R) \). For a substitution \( P \) and a set of type variables \( V \), we write \( P|_V \) for \( \{ \tau/\alpha \in P \mid \alpha \notin V \} \). The notation \( \forall \vec{\alpha}. \tau' \succ \tau \) means that there exists a substitution \( S \) such that \( S\tau' = \tau \) and \( supp(S) \subseteq \vec{\alpha} \). We write \( \Gamma + x: \sigma \) to mean \( \{ y \mapsto \sigma' \mid x \neq y, y \mapsto \sigma' \in \Gamma \} \cup \{ x \mapsto \sigma \} \). \( Clos_{\Gamma}(\tau) \) is the same as \( Gen(\Gamma, \tau) \) in [DM82], i.e., \( \forall \vec{\alpha}. \tau \), where \( \vec{\alpha} = ftv(\tau) \setminus ftv(\Gamma) \).

2 The Generalized Algorithm \( \mathcal{G} \)

2.1 Overview

The source language and its Hindley/Milner style let-polymorphic type system are shown in Figure 1. The two opposite algorithms \( W \) and \( M \) are shown in Figure 2.

Our generalized algorithm is based on the top-down, context-sensitive algorithm \( M \). Key observation is that varying the type-checking strategy is possible by changing two factors in \( M \): the information amount of the type constraints to pass to type-checking sub-expressions and the places of the unification. Algorithm \( M \) carries as much information as possible at its type constraints and applies a unification at every value (constant, variable, and lambda) expression. Algorithm \( W \), on the other hand, carries no information at its type constraints and applies a unification at every application expression. By tuning the two factors, other type-checking strategies are also possible:

Example 1 Consider an application expression

\( \text{(IsOne 2)} : \text{bool} \)
Abstract Syntax

\[ \text{Expr} \quad e \ ::= \begin{cases} () & \text{constant} \\ x & \text{variable} \\ \lambda x.e & \text{function} \\ e \ e & \text{application} \\ \text{let } x = e \text{ in } e \\ \text{fix } f \lambda x.e \end{cases} \]

\[ \text{Type} \quad \tau \ ::= \begin{cases} \iota & \text{constant type} \\ \alpha & \text{type variable} \\ \tau \rightarrow \tau & \text{function type} \end{cases} \]

\[ \text{TypeScheme} \quad \sigma \ ::= \begin{cases} \tau & \text{type scheme} \\ \forall \vec{\alpha}.\sigma & \text{function type} \end{cases} \]

\[ \text{TypeEnv} \quad \Gamma \in Var^\rightarrow TypeScheme \quad \text{type environment} \]

\[
\begin{align*}
\text{(CON)} & \quad \Gamma \vdash () : \iota \\
\text{(VAR)} & \quad \Gamma(x) : \tau \\
\text{(FN)} & \quad \Gamma \vdash x : \tau \\
\text{(APP)} & \quad \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1 \\
\text{(LET)} & \quad \Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2 \\
\text{(FIX)} & \quad \Gamma \vdash \text{fix } f \lambda x.e : \tau
\end{align*}
\]

Figure 1: Language and its let-polymorphic type system

where \textit{IsOne} has type \texttt{int} \rightarrow \texttt{bool}. As we impose less and less constraints in type-checking sub-expressions yet apply more and more checks later, we achieve the following type-checking variations:

- We type-check \textit{IsOne} with constraint \(\beta \rightarrow \texttt{bool}\), which is the strongest expectation. After its success, we type-check 2 with the function’s domain type \texttt{int} as its constraint. (M)

- We type-check \textit{IsOne} with a weaker constraint, \(\beta_1 \rightarrow \beta_2\) with \(\beta_1\) and \(\beta_2\) being new type variables. The constraint enforces that \textit{IsOne}’s type be just a function, whatever its domain and range types are. After its success, we check whether the function’s range type is \texttt{bool}. Then we type-check 2 with the function’s domain type \texttt{int} as its constraint.

- We type-check \textit{IsOne} with no constraint. After its success, we check whether the result type is a function type to \texttt{bool}. Then we type-check 2 with the function’s domain type \texttt{int} as its constraint. (OCaml’s type inference algorithm)

- We type-check \textit{IsOne} with no constraint. After its success, we check whether the result type is just a function type, whatever its domain and range types are. Then we type-check 2 with the function’s domain type \texttt{int} as its constraint. After its success, we check whether the function’s range type is \texttt{bool}. 
\[ W: \text{TypEnv} \times \text{Expr} \rightarrow \text{Subst} \times \text{Type} \]

\[ W(\Gamma, \epsilon) = (\text{id}, \epsilon) \]  \hspace{1cm} (W.1)

\[ W(\Gamma, x) = (\text{id}, \{\beta/\alpha\}\tau) \text{ where } \Gamma(x) = \forall \alpha.\tau, \text{ new } \beta \]  \hspace{1cm} (W.2)

\[ W(\Gamma, \lambda x.e) = \text{let } (S_1, \tau_1) = W(\Gamma + x: \beta, e), \text{ new } \beta \]  \hspace{1cm} (W.3)

\[ W(\Gamma, e_1 e_2) = \text{let } (S_1, \tau_1) = W(\Gamma, e_1) \]
\[ (S_2, \tau_2) = W(S_1 \Gamma, e_2) \]  \hspace{1cm} (W.4)

\[ S_3 = U(S_2 \tau_1, \tau_2 \rightarrow \beta), \text{ new } \beta \]  \hspace{1cm} (W.5)

\[ \text{in } (S_3 S_2 S_1, S_3 \beta) \]

\[ W(\Gamma, \text{let } x = e_1 \text{ in } e_2) = \]

\[ \text{let } (S_1, \tau_1) = W(\Gamma, e_1) \]
\[ (S_2, \tau_2) = W(S_1 \Gamma + x: \text{Class} S_1 r(\tau_1), e_2) \]  \hspace{1cm} (W.7)

\[ \text{in } (S_2 S_1, \tau_2) \]

\[ W(\Gamma, \text{fix } f \lambda x.e) = \text{let } (S_1, \tau_1) = W(\Gamma + f: \beta, \lambda x.e), \text{ new } \beta \]
\[ S_2 = U(S_1 \beta, \tau_1) \]  \hspace{1cm} (W.9)

\[ \text{in } (S_2 S_1, S_2 \tau_1) \]

\[ \mathcal{M}: \text{TypEnv} \times \text{Expr} \times \text{Type} \rightarrow \text{Subst} \]

\[ \mathcal{M}(\Gamma, \epsilon, \rho) = U(\rho, \epsilon) \]  \hspace{1cm} (M.1)

\[ \mathcal{M}(\Gamma, x, \rho) = U(\rho, \{\beta/\alpha\}\tau) \text{ where } \Gamma(x) = \forall \alpha.\tau, \text{ new } \beta \]  \hspace{1cm} (M.2)

\[ \mathcal{M}(\Gamma, \lambda x.e, \rho) = \text{let } S_1 = U(\rho, \beta_1 \rightarrow \beta_2), \text{ new } \beta_1, \beta_2 \]
\[ S_2 = M(S_1 \Gamma + x: S_1 \beta_1, e, S_1 \beta_2) \]  \hspace{1cm} (M.3)

\[ \text{in } S_2 S_1 \]

\[ \mathcal{M}(\Gamma, e_1 e_2, \rho) = \text{let } S_1 = M(\Gamma, e_1, \beta \rightarrow \rho), \text{ new } \beta \]
\[ S_2 = M(S_1 \Gamma, e_2, S_1 \beta) \]  \hspace{1cm} (M.5)

\[ \text{in } S_2 S_1 \]

\[ \mathcal{M}(\Gamma, \text{let } x = e_1 \text{ in } e_2, \rho) = \]

\[ \text{let } S_1 = M(\Gamma, e_1, \beta), \text{ new } \beta \]
\[ S_2 = M(S_1 \Gamma + x: \text{Class} S_1 r(S_1 \beta), e_2, S_1 \rho) \]  \hspace{1cm} (M.7)

\[ \text{in } S_2 S_1 \]

\[ \mathcal{M}(\Gamma, \text{fix } f \lambda x.e, \rho) = \mathcal{M}(\Gamma + f: \rho, \lambda x.e, \rho) \]  \hspace{1cm} (M.9)

Figure 2: The definition of \( W \) and \( \mathcal{M} \). Every new type variable is distinct from each other, and the set New of new type variables introduced at each recursive call to \( W(\Gamma, e) \) (respectively, \( \mathcal{M}(\Gamma, e, \rho) \)) satisfies \( \text{New} \cap \text{fvar}(\Gamma) = \emptyset \) (respectively, \( \text{New} \cap (\text{fvar}(\Gamma) \cup \text{fvar}(\rho)) = \emptyset \).)

- We type-check \texttt{IsOne} with no constraint. After its success, we check, as before, whether the result type is just a function type. Then we type-check 2, but with no constraint. After its success, we check whether the function’s type is \texttt{int \rightarrow bool}. (W)

- We type-check \texttt{IsOne} with no constraint. After its success, we don’t check anything but continue type-checking the second expression 2 with no constraint. After its success, we check everything at once: we check whether \texttt{IsOne}’s type is a function type from \texttt{int} to \texttt{bool}. (W) \hfill \square

Every type-checking variation in the above example exposes a common property: it loosens the type constraints for sub-expressions then checks afterward whether the results from loosened constraints agree with the contexts implied from the original, unloosened constraints.

Our generalized algorithm is one that allows, wherever possible, the loosening of the type
for the loosened constraint used in type-checking each sub-expression. The unifications check constraints at the two recursive calls. This is done by two unifications: each one compensates sub-expressions to type-check, hence it’s time to finalize the compensation for the loosened implies.

The places for loosening the constraints are right before recursive calls for type-checking sub-expressions. The places for posterior unifications that compensate for the loosened constraints are after the successful returns from the recursive-calls. Some unifications may only partially compensate for the loosened constraints. Thus, before the original call returns there must be final unification(s) that completes the compensations. For example, consider type-checking application expression $e_1 e_2$ with initial constraint $\rho$. It type-checks $e_1$ with a type constraint that can be less restraining than the strongest possible constraint $\beta \rightarrow \rho$. Right after its return, it applies a unification that can compensate, not necessarily completely, for the loosened constraint. It then type-checks the argument expression $e_2$ with a type constraint that can be less restraining than the type that the $\beta$ became. After its success, there exists no more sub-expressions to type-check, hence it’s time to finalize the compensation for the loosened constraints at the two recursive calls. This is done by two unifications: each one compensates for the loosened constraint used in type-checking each sub-expression. The unifications check whether the types from the loosened constraints agree with what the strongest constraint $\beta \rightarrow \rho$ implies.

Figure 3: Generalized type inference algorithm $\mathcal{G}$. All the type variables in $\text{ftv}(\theta) \setminus \text{ftv}(\rho)$ (respectively, $(\text{ftv}(\theta) \cup \text{ftv}(\rho)) \setminus \text{ftv}(\rho)$) for each $\theta \geq \rho$ (respectively, $\theta_1 \land \theta_2 \geq \rho$) are new, every new type variable is distinct from each other, and the set $\text{New}$ of new type variables introduced at each recursive call to $\mathcal{G}(\Gamma, e, \rho)$ satisfies $\text{New} \cap (\text{ftv}(\Gamma) \cup \text{ftv}(\rho)) = \emptyset$.  

<table>
<thead>
<tr>
<th>$\mathcal{G}$ : $\mathcal{G}$</th>
<th>$\mathcal{G}$</th>
<th>$\mathcal{G}$</th>
<th>$\mathcal{G}$</th>
<th>$\mathcal{G}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{G}(\Gamma, \text{O}, \rho) = \mathcal{U}(\rho, \nu)$</td>
<td>(G.1)</td>
<td>(G.1)</td>
<td>(G.1)</td>
<td>(G.1)</td>
</tr>
<tr>
<td>$\mathcal{G}(\Gamma, x, \rho) = \mathcal{U}(\rho, {\beta/\alpha} \tau)$, new $\beta$, $\Gamma(x) = \forall \alpha. \tau$</td>
<td>(G.2)</td>
<td>(G.2)</td>
<td>(G.2)</td>
<td>(G.2)</td>
</tr>
<tr>
<td>$\mathcal{G}(\Gamma, \lambda x. e, \rho) =$</td>
<td>let $S_1 = \mathcal{U}(\beta_1 \rightarrow \beta_2, \theta)$, new $\beta_1$, new $\beta_2, \theta \geq \rho$</td>
<td>(G.3)</td>
<td>(G.3)</td>
<td>(G.3)</td>
</tr>
<tr>
<td>$S_2 = \mathcal{G}(S_1 \Gamma + x: S_1 \beta_1, e, S_1 \beta_2)$</td>
<td>(G.4)</td>
<td>(G.4)</td>
<td>(G.4)</td>
<td>(G.4)</td>
</tr>
<tr>
<td>$S_3 = \mathcal{U}(S_2 S_1 \theta, S_2 S_1 \rho)$</td>
<td>(G.5)</td>
<td>(G.5)</td>
<td>(G.5)</td>
<td>(G.5)</td>
</tr>
<tr>
<td>in $S_1 S_2 S_1$</td>
<td></td>
<td>(G.6)</td>
<td>(G.6)</td>
<td>(G.6)</td>
</tr>
<tr>
<td>$\mathcal{G}(\Gamma, e_1 e_2, \rho) =$</td>
<td>let $S_1 = \mathcal{G}(\Gamma, e_1, \theta_1)$, new $\beta$, $\theta_1 \geq \beta \rightarrow \rho$</td>
<td>(G.7)</td>
<td>(G.7)</td>
<td>(G.7)</td>
</tr>
<tr>
<td>$S_2 = \mathcal{U}(S_1 \theta_1, \theta_2)$, $\theta_2 \geq S_1(\beta \rightarrow \rho)$</td>
<td>(G.8)</td>
<td>(G.8)</td>
<td>(G.8)</td>
<td>(G.8)</td>
</tr>
<tr>
<td>$S_3 = \mathcal{G}(S_2 S_1 \Gamma, e_2, \theta_3)$, $\theta_3 \geq S_2 S_1 \beta$</td>
<td>(G.9)</td>
<td>(G.9)</td>
<td>(G.9)</td>
<td>(G.9)</td>
</tr>
<tr>
<td>$S_4 = \mathcal{U}(S_3 S_2 S_1 \theta_1, S_3 S_2 S_1 (\beta \rightarrow \rho))$</td>
<td>(G.10)</td>
<td>(G.10)</td>
<td>(G.10)</td>
<td>(G.10)</td>
</tr>
<tr>
<td>in $S_1 S_2 S_1 S_3 S_2 S_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathcal{G}(\Gamma, \text{let } x = e_1 \text{ in } e_2, \rho) =$</td>
<td>let $S_1 = \mathcal{G}(\Gamma, e_1, \beta)$, new $\beta$</td>
<td>(G.11)</td>
<td>(G.11)</td>
<td>(G.11)</td>
</tr>
<tr>
<td>$S_2 = \mathcal{G}(S_1 \Gamma + x: \text{Clos}_{S_1}(S_1 \beta), e_2, \theta)$, $\beta \rightarrow \rho$</td>
<td>(G.12)</td>
<td>(G.12)</td>
<td>(G.12)</td>
<td>(G.12)</td>
</tr>
<tr>
<td>$S_3 = \mathcal{U}(S_2 \theta, S_2 S_1 \rho)$</td>
<td>(G.13)</td>
<td>(G.13)</td>
<td>(G.13)</td>
<td>(G.13)</td>
</tr>
<tr>
<td>in $S_1 S_2 S_1$</td>
<td></td>
<td>(G.14)</td>
<td>(G.14)</td>
<td>(G.14)</td>
</tr>
<tr>
<td>$\mathcal{G}(\Gamma, \text{fix } f \lambda x. e, \rho) =$</td>
<td>let $S_1 = \mathcal{G}(\Gamma + f; \theta_1, \lambda x. e, \theta_2)$, $\theta_1 \land \theta_2 \geq \rho$</td>
<td>(G.15)</td>
<td>(G.15)</td>
<td>(G.15)</td>
</tr>
<tr>
<td>$S_2 = \mathcal{U}(S_1 \theta_1, S_1 \theta_2, S_1 \rho)$</td>
<td>(G.16)</td>
<td>(G.16)</td>
<td>(G.16)</td>
<td>(G.16)</td>
</tr>
<tr>
<td>in $S_2 S_1$</td>
<td></td>
<td>(G.17)</td>
<td>(G.17)</td>
<td>(G.17)</td>
</tr>
</tbody>
</table>
2.2 Algorithm Definition

The generalized algorithm \( G \) is shown in Figure 3. As in \( M \), it returns a substitution from three components: an expression, a type environment, and a type constraint. The inferred type of the expression is the result from applying the final substitution to the type constraint of the expression. The type constraints are just types.

By the phrases of the form \( \theta \geq \rho \) marked (1) to (6) in the algorithm, the most strongest type constraint \( \rho \) is loosened into \( \theta \) at each recursive call. This less restraining type constraint is the one that can be instantiated to \( \rho \) by a substitution that ranges over the type variables in only \( \theta \):

**Definition 1** \(( \theta \geq \rho )\) Type \( \theta \) is more general (less restraining) than type \( \rho \), written \( \theta \geq \rho \), if and only if there exists a substitution \( G \) such that \( G\theta = \rho \) and \( \text{supp}(G) = \text{ftv}(\theta) \setminus \text{ftv}(\rho) \). We write \( \theta_1 \land \theta_2 \geq \rho \) if and only if there exists a substitution \( G \) such that \( G\theta_1 = \rho \) and \( G\theta_2 = \rho \) and \( \text{supp}(G) = (\text{ftv}(\theta_1) \cup \text{ftv}(\theta_2)) \setminus \text{ftv}(\rho) \).

For the variable case \((G.2)\), the variable’s type \( \Gamma(x) \) must satisfy the current type constraint \( \rho \): \( \mathcal{U}(\rho, [\vec{\beta}/\vec{\alpha}]\tau) \). Similarly for the constant case \((G.1)\).

For the lambda expression case \( \lambda x.e \) with type constraint \( \rho \), we first decide on the type constraint for the function’s body expression \( e \). It can be any type that is less restraining than the range type of \( \rho \). We choose such a type by loosening \( \rho \) first, then picking up its range component by unification:

\[
S_1 = \mathcal{U}(\beta_1 \rightarrow \beta_2, \theta), \text{ new } \beta_1, \beta_2, \quad (1) \theta \geq \rho. \tag{G.3}
\]

Then we use the resulting range type \( S_1\beta_2 \) as the constraint in type-checking the function’s body expression:

\[
S_2 = G(S_1\Gamma + x, S_1\beta_1, e, S_1\beta_2). \tag{G.4}
\]

For example, if we choose the \( \theta \) to be a new type variable, then the unification \((G.3)\) has no effect, hence \( e \)’s type is inferred without any constraint. The other extreme is to choose \( \theta \) to be the \( \rho \). Then \( e \)’s type is inferred with \( \rho \)’s range type, if \( \rho \) is a function type.

After returning from the recursive call to \( e \), we have to make up for passing less restraining type constraint. This last step is done by checking whether the loosened constraint \( \theta \) can agree with the type that its original \( \rho \) becomes:

\[
S_3 = \mathcal{U}(S_2S_1\theta, S_2S_1\rho). \tag{G.5}
\]

Consider type-checking application expression \( e_1 e_2 \) with type constraint \( \rho \). First we decide on the type constraint for the function expression \( e_1 \). It can be any type that is less restraining than the most informative constraint \( \beta \rightarrow \rho \) with \( \beta \) being a new type variable:

\[
S_1 = G(\Gamma, e, \theta_1), \text{ new } \beta, \quad (2) \theta_1 \geq \beta \rightarrow \rho. \tag{G.6}
\]

After the success of this recursive call and before we continue by type-checking the argument expression, we can make up, not necessarily completely, for passing less restraining type constraint \( \theta_1 \). This reparation can be varied by how much we want to expect for the type of \( e_1 \). We can check the result type against the strongest constraint \( \beta \rightarrow \rho \) or we can check against nothing. This varied degree of reparation is achieved by choosing yet another less restraining type \( \theta_2 \) than \( S_1(\beta \rightarrow \rho) \) and by unifying it with the type that \( \theta_1 \) becomes:
\[ S_2 = \mathcal{U}(S_1 \theta_1, \theta_2), \quad (3) \quad \theta_2 \geq S_1(\beta \rightarrow \rho). \]  \hfill (G.7)

Next we decide on the type constraint to pass for type-checking the argument expression \( e_2 \). It can be any type that is less constraining than the type that \( \beta \) becomes. Hence the next recursive call is:

\[ S_3 = \mathcal{G}(S_2 S_1 \Gamma, e_2, \theta_3), \quad (4) \quad \theta_3 \geq S_2 S_1 \beta. \]  \hfill (G.8)

The finalizing compensation for passing the less constraining type constraints to the two recursive calls are done by checking whether the first loosened constraint \( \theta_1 \) can agree with the type that the original type \( \beta \rightarrow \rho \) becomes:

\[ S_4 = \mathcal{U}(S_3 S_2 S_1 \theta_1, S_3 S_2 S_1(\beta \rightarrow \rho)) \]  \hfill (G.9)

and by checking whether the other loosened constraint \( \theta_3 \) for the argument expression can agree with what the original type \( \beta \) becomes:

\[ S_5 = \mathcal{U}(S_4 S_3 \theta_3, S_4 S_3 S_2 S_1 \beta). \]  \hfill (G.10)

We don’t have to check for \( \theta_2 \) because of its unification with \( \theta_1 \) at line (G.7).

Consider inferring the type of let-expression \( \text{let } x = e_1 \text{ in } e_2 \) with type constraint \( \rho \). Because there is no context information about the type of the first expression \( e_1 \), there is no room for varying its type constraint:

\[ S_1 = \mathcal{G}(\Gamma, e_1, \beta), \text{ new } \beta. \]  \hfill (G.11)

Next we decide on the type constraint for the body expression \( e_2 \). It can be any type that is less constraining than the given constraint \( \rho \):

\[ S_2 = \mathcal{G}(S_1 \Gamma + x : \text{Close}_{S_1 \Gamma}(S_1 \beta), e_2, \theta), \quad (5) \quad \theta \geq S_1 \rho. \]  \hfill (G.12)

Finally, we have to check whether the loosened type agrees with the type that the original constraint becomes:

\[ S_3 = \mathcal{U}(S_2 \theta, S_2 S_1 \rho). \]  \hfill (G.13)

The case for recursive function \( \text{fix } f \lambda x.e \) is similar. We decide on what is expected for the type of \( \lambda x.e \) and what is carried for the type of \( f \). Both can be less constraining than \( \rho \):

\[ S_1 = \mathcal{G}(\Gamma + f : \theta_1, \lambda x.e, \theta_2), \quad (6) \quad \theta_1 \wedge \theta_2 \geq \rho. \]  \hfill (G.14)

Then we check whether the loosened type agrees with the type that the original constraint becomes:

\[ S_2 = \mathcal{U}(S_1 \theta_1, S_1 \theta_2, S_1 \rho). \]  \hfill (G.15)
2.3 Instances

By determining the loosened constraints $\theta$’s in $G$, we obtain various type-inference algorithms, including the standard Algorithm $W$, the folklore top-down algorithm $M$, and the combinations of the two algorithms used in the SML/NJ [SL99] and OCaml [LRVD99] compiler systems.

- $W$ is an instance of $G$ where every $\theta$ is a new type variable.
- $M$ is an instance of $G$ where every $\theta$ is not loosened: for each case $\theta \geq \rho$ in $G$, we choose $\rho$ for $\theta$.
- The OCaml’s type inference algorithm is an instance of $G$ where the $\theta$ at (2) (line (G.6)) is a new type variable and other $\theta$’s are not loosened.
- The SML/NJ’s type inference algorithm is an instance of $G$ where the $\theta$ at (1) (line (G.3)) is $\rho$ if the lambda is a recursive function, otherwise, a new type variable, the $\theta_1$ and $\theta_2$ at (6) (line (G.14)) are the same new type variable, and other $\theta$’s are new type variables.
- Other variations than the existing algorithms are also possible from $G$. For example, consider an instance of $G$ where the $\theta$ at (G.6) is a new function type ($\beta_1 \to \beta_2$ for new variables $\beta_1$ and $\beta_2$) and other $\theta$’s are their most restraining constraints. Let’s call this instance algorithm $H$.

The $\theta$’s used in the five instances are summarized in Figure 4.

3 Every Instance Is Sound and Complete

Every instance of $G$ is sound and complete with respect to the Hindley/Milner let-polymorphic type system.

**Theorem 1 (Soundness)** Let $e$ be an expression, $\Gamma$ be a type environment, and $\rho$ be a type. If $G(\Gamma, e, \rho)$ succeeds with $S$, then $ST \vdash e : S\rho$.

The proof uses Lemmas 1 to 3 and Theorem 2.

**Lemma 1** [DM82] If $\Gamma \vdash e : \tau$, then $ST \vdash e : S\tau$.

**Lemma 2** [DM82] If $\sigma \succ \sigma'$ then $S\sigma \succ S\sigma'$.

**Lemma 3** [Mil78] Let $S$ be a substitution, $\Gamma$ be a type environment, and $\tau$ be a type. $SClos_T(\tau) = Clos_{ST}(S'\tau)$, where $S' = S(\vec{\beta}/\vec{\alpha})$, $\vec{\alpha} = ftv(\tau) \setminus ftv(\Gamma)$ and $\vec{\beta}$ is new.
Theorem 2 [Rob65] There is an algorithm $\mathcal{U}$ which, given a pair of types, either returns a substitution $S$ or fails; further

- If $S = \mathcal{U}(\tau, \tau')$ then $S\tau = S\tau'$.
- If $S'$ unifies $\tau$ and $\tau'$, then $\mathcal{U}(\tau, \tau')$ succeeds with $S$ and there exists a substitution $R$ such that $S' = RS$.

Moreover, $S$ involves only variables of $\tau$ and $\tau'$.

Proof of Theorem 1. We prove by structural induction on $e$.

- **case $\emptyset$:** $S\rho = S\iota = \iota$. So $S\Gamma \vdash \emptyset : S\rho$ by (CON).
- **case $x$:** $S\rho = S(\beta/\alpha)\tau < ST(x)$ by Lemma 2. So $S\Gamma \vdash x : S\rho$ by (VAR).
- **case $\lambda x.e$:** By induction hypothesis, $(\mathcal{G}.4)$ implies that $S_2S_1\Gamma + x : S_2S_1\beta_1 \vdash e : S_2S_1\beta_2$.

  By Lemma 1, we can apply $S_3$ to both sides:

  $$S_3S_2S_1\Gamma \vdash \lambda x.e : S_2S_1(\beta_1 \rightarrow \beta_2).$$

  Because $S_1(\beta_1 \rightarrow \beta_2) = S_1\theta$ by $(\mathcal{G}.3)$ and $S_3S_2S_1\theta = S_3S_2S_1\rho$ by $(\mathcal{G}.5)$,

  $$S_3S_2S_1(\beta_1 \rightarrow \beta_2) \vdash \lambda x.e : S_3S_2S_1(\beta_1 \rightarrow \beta_2).$$

- **case $e_1, e_2$:** By induction, $(\mathcal{G}.6)$ implies $S_1\Gamma \vdash e_1 : S_1\theta_1$. By Lemma 1, we can apply $S_5S_4S_3S_2$ to both sides:

  $$S_5S_4S_3S_2S_1\Gamma \vdash e_1 : S_5S_4S_3S_2S_1\theta_1.$$

  Because $S_3S_2S_1\theta_1 = S_3S_2S_1(\beta \rightarrow \rho)$ by $(\mathcal{G}.9)$ and $S_5S_4S_3S_2S_1\beta = S_5S_4S_3S_2\beta_3$ by $(\mathcal{G}.10)$,

  $$S_5S_4S_3S_2S_1\Gamma \vdash e_1 : S_5S_4S_3(\beta_3 \rightarrow S_2S_1\rho).$$

  By induction, $(\mathcal{G}.8)$ implies $S_3S_2S_1\Gamma \vdash e_2 : S_3\theta_3$. By Lemma 1, we can apply $S_5S_4$ to both sides:

  $$S_5S_4S_3S_2S_1\Gamma \vdash e_2 : S_5S_4S_3\theta_3.$$  

  Hence by (APP), (7) and (8) imply

  $$S_5S_4S_3S_2S_1\Gamma \vdash e_1 e_2 : S_5S_4S_3S_2S_1\rho.$$

- **case $\mathtt{let} \ x = e_1 \ \mathtt{in} \ e_2$:** Let $S'_1 = S_2(\beta/\alpha)$, where $\alpha = \text{fte}(S_1\beta) \setminus \text{fte}(S_1\Gamma)$, $\beta$ are new type variables, and $\beta$ is the new type variable introduced at $(\mathcal{G}.11)$. By induction, $(\mathcal{G}.11)$ implies $S_1\Gamma \vdash e_1 : S_1\beta$. By Lemma 1, we can apply $S'_2$ to both sides:

  $$S'_2S_1\Gamma \vdash e_1 : S'_2S_1\beta.$$  

  By induction, $(\mathcal{G}.12)$ implies

  $$S_2S_1\Gamma + x : S_2\text{Close}_{S_1}(S_1\beta) \vdash e_2 : S_2\theta.$$  


Note that $S_2S_1\Gamma = S'_2S_1\Gamma$ because $S'_2$ differs from $S_2$ only on non-free variables of $S_1\Gamma$, and that $S_2\text{Class}_{\Gamma}(S_1\beta) = Clos_{S_2S_1\Gamma}(S'_2S_1\beta)$ by Lemma 3. Thus (10) is

$$S'_2S_1\Gamma + x : Clos_{S'_2S_1\Gamma}(S'_2S_1\beta) \vdash e_2 : S_2\theta.$$  \hspace{1cm} (11)

Hence by (LET), (9) and (11) imply $S'_2S_1\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : S_2\theta$; that is,

$$S_2S_1\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : S_2\theta.$$

By Lemma 1, we can apply $S_3$ to both sides:

$$S_3S_2S_1\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : S_3S_2\theta.$$

Because $S_3S_2\theta = S_3S_2S_1\rho$ by (G.13),

$$S_3S_2S_1\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : S_3S_2S_1\rho.$$

- **case** $\text{fix } f \lambda x.e$: By induction, (G.14) implies $S_1\Gamma + f : S_1\theta_1 \vdash \lambda x.e : S_1\theta_2$. By Lemma 1, we can apply $S_2$ to both sides:

$$S_2S_1\Gamma + f : S_2S_1\theta_1 \vdash \lambda x.e : S_2S_1\theta_2.$$

Because $S_2S_1\theta_1 = S_2S_1\theta_2 = S_2S_1\rho$ by (G.15),

$$S_2S_1\Gamma + f : S_2S_1\rho \vdash \lambda x.e : S_2S_1\rho.$$

Hence by (FIX),

$$S_2S_1\Gamma \vdash \text{fix } f \lambda x.e : S_2S_1\rho. \quad \Box$$

**Theorem 3 (Completeness)** Let $e$ be an expression, and let $\Gamma$ be a type environment. If there exist a type $\rho$ and a substitution $P$ such that $P\Gamma \vdash e : P\rho$, then $G(\Gamma, e, \rho)$ succeeds with $S$ and there exists a substitution $R$ such that $P|_{\text{New}} = (RS)|_{\text{New}}$ where New is the set of new type variables used by $G(\Gamma, e, \rho)$.

Completeness means that if an expression $e$ has a type $\tau$ that satisfies a type constraint $\rho$ (i.e., $\exists P. \tau = P\rho$), then algorithm $G$ for the expression with the constraint $\rho$ succeeds with substitution $S$ such that the result type $S\rho$ subsumes $\tau$ (i.e., the principality, $\exists R. \tau = R(S\rho)$). The completeness proof uses Lemmas 4 to 8.

**Lemma 4** [LY98] Let $S$ be a substitution, $\Gamma$ be a type environment, and $\tau$ be a type. Then $S\text{Class}(\tau) \supset Clos_{S\Gamma}(\tau)$.

**Lemma 5** [DM82] Let $\Gamma$ and $\Gamma'$ be type environments such that $\Gamma \supset \Gamma'$. If $\Gamma' \vdash e : \tau$, then $\Gamma \vdash e : \tau$.

**Lemma 6** [Mil78] Let $R$ and $S$ be substitutions and $\tau$ be a type. Then

- $\text{itv}(RS) \subseteq \text{itv}(R) \cup \text{itv}(S)$ and

- $\text{ftv}(S\tau) \subseteq \text{ftv}(\tau) \cup \text{itv}(S)$.

**Lemma 7** If $S = G(\Gamma, e, \rho)$ then $\text{itv}(S) \subseteq \text{ftv}(\Gamma) \cup \text{ftv}(\rho) \cup \text{New}$, where New is the set of new type variables used by $G(\Gamma, e, \rho)$.
Proof. See Appendix A. □

**Lemma 8** [LY98] If itv(S) \( \cap V = \emptyset \), then \((RS)|_V = R|_V S\).

**Proof of Theorem 3.** We prove by structural induction on \( e \). For a rigorous treatment of new type variables, we assume that every new type variable used throughout algorithm \( G \) is distinct from each other, and that the set \( \text{New} \) of new type variables used by each call \( G(\Gamma, e, \rho) \) satisfies \( \text{New} \cap (\text{ftv}(\Gamma) \cup \text{ftv}(\rho)) = \emptyset \). Moreover, let us rephrase the part of the algorithm definition that whenever we use \( \theta \) and has on only new type variables.

- **case () and x:** The same as the proof for \( M \) in [LY98].
- **case \( \lambda x.e \):** Let the given judgment be \( P \Gamma \vdash \lambda x.e : \tau_1 \rightarrow \tau_2 \) where \( \tau_1 \rightarrow \tau_2 = P \rho \), and \( \text{New} = \{ \beta_1, \beta_2 \} \cup \text{supp}(G) \cup \text{New}_1 \) where \( \beta_1 \) and \( \beta_2 \) are new type variables used at \((G.3)\), \( G \) is the substitution for \( \theta \geq \rho \) at \((G.3)\), and \( \text{New}_1 \) is the set of new type variables used by \( G(S_1 \Gamma + x: S_1 \beta_1, e, S_1 \beta_2) \) at \((G.4)\).

First, we prove the unification \( U(\beta_1 \rightarrow \beta_2, \theta) \) at \((G.3)\) succeeds. Let \( P' = (PG)|_{(\beta_1, \beta_2)} \cup \{ \tau_1/\beta_1, \tau_2/\beta_2 \} \). Then \( P' \) unifies \( \beta_1 \rightarrow \beta_2 \) and \( \theta \) because

\[
P' \theta = PG\theta \quad \text{because the new } \beta_1, \beta_2 \notin \text{ftv}(\theta)
\]

\[
= P\rho \quad \text{by the definition of } G
\]

\[
= \tau_1 \rightarrow \tau_2 \quad \text{by the assumption}
\]

\[
= P'(\beta_1 \rightarrow \beta_2) \quad \text{by the definition of } P'.
\]

Thus by Theorem 2, the unification at \((G.3)\) succeeds with \( S_1 \) such that for a substitution \( R_1 \),

\[
R_1 S_1 = P'.
\] (12)

By the \((FN)\) rule, the given judgment implies

\[
P \Gamma + x: \tau_1 \vdash e : \tau_2.
\] (13)

To apply induction to \( G(S_1 \Gamma + x: S_1 \beta_1, e, S_1 \beta_2) \) at \((G.4)\) and \((13)\), we must prove that there exists a substitution \( P_1 \) such that \( \tau_2 = P_1(S_1 \beta_2) \) and \( P \Gamma + x: \tau_1 = P_1(S_1 \Gamma + x: S_1 \beta_1) \). Such \( P_1 \) is \( R_1 \) at \((12)\) because

\[
R_1(S_1 \beta_2) = P'\beta_2 \quad \text{by } (12)
\]

\[
= \tau_2 \quad \text{by the definition of } P'
\]

and

\[
R_1(S_1 \Gamma + x: S_1 \beta_1) = P'(\Gamma + x: \beta_1) \quad \text{by } (12)
\]

\[
= PG\Gamma + x: \tau_1 \quad \text{because the new } \beta_1, \beta_2 \notin \text{ftv}(\Gamma)
\]

\[
= P \Gamma + x: \tau_1 \quad \text{because } \text{supp}(G) \cap \text{ftv}(\Gamma) = \emptyset.
\]

Thus by induction, \( G(S_1 \Gamma + x: S_1 \beta_1, e, S_1 \beta_2) \) at \((G.4)\) succeeds with \( S_2 \) such that for a substitution \( R_2 \),

\[
(R_2 S_2)|_{\text{New}_1} = R_1|_{\text{New}_1}.
\] (14)
March 31, 2000

ROPAS-2000-5

Note that
\[
\text{ftv}(S_1) 
\subseteq \{\beta_1, \beta_2\} \cup \text{ftv}(\theta) \quad \text{by Theorem 2}
\]
\[
\subseteq \{\beta_1, \beta_2\} \cup \text{ftv}(\rho) \cup \text{supp}(G) \quad \text{because supp}(G) = \text{ftv}(\theta) \setminus \text{ftv}(\rho)
\]

and thus by the definition of \( G \),
\[
\text{New}_1 \cap \text{ftv}(S_1) = \emptyset. \quad (15)
\]

Then
\[
(R_2S_2S_1)\mid_{\text{New}_1} = (R_2S_2)\mid_{\text{New}_1}S_1 \quad \text{by Lemma 8 and (15)}
\]
\[
= R_1\mid_{\text{New}_1}S_1 \quad \text{by (14)}
\]
\[
= (R_1S_1)\mid_{\text{New}_1} \quad \text{by Lemma 8 and (15)}
\]
\[
= P^\prime_1\mid_{\text{New}_1} \quad \text{by (12)}. \quad (16)
\]

Now we prove the unification \( \mathcal{U}(S_2S_1, S_2S_1\rho) \) at (G.5) succeeds. \( R_2 \) unifies \( S_2S_1\theta \) and \( S_2S_1\rho \) because
\[
R_2(S_2S_1) = P^\prime \theta \quad \text{by (16) and because ftv(\theta) \cap \text{New}_1 = \emptyset}
\]
\[
= PG\theta \quad \text{because the new } \beta_1, \beta_2 \notin \text{ftv}(\theta)
\]
\[
= P\rho \quad \text{by the definition of } G
\]
\[
= PG\rho \quad \text{because ftv(\rho) \cap supp}(G) = \emptyset
\]
\[
= P^\prime\rho \quad \text{because the new } \beta_1, \beta_2 \notin \text{ftv}(\rho)
\]
\[
= R_2(S_2S_1\rho) \quad \text{by (16) and because ftv(\rho) \cap \text{New}_1 = \emptyset}.
\]

Thus the unification at (G.5) succeeds with \( S_3 \) such that for a substitution \( R_3 \),
\[
R_3S_3 = R_2. \quad (17)
\]

Hence \( G(\Gamma, \lambda x.e, \rho) \) succeeds with \( S_3S_2S_1, \) and \( (R_3S_3S_2S_1)\mid_{\text{New}} = P\mid_{\text{New}} \) because
\[
(R_3S_3S_2S_1)\mid_{\text{New}} = (R_2S_2S_1)\mid_{\text{New}} \quad \text{by (17)}
\]
\[
= P^\prime\mid_{\text{New}} \quad \text{by (16)}
\]
\[
= P\mid_{\text{New}} \quad \text{because supp}(G) \cup \{\beta_1, \beta_2\} \subseteq \text{New}.
\]

- **case** \( e_1, e_2 \): Let the given judgment be \( PT \vdash e_1, e_2 : P\rho, \) and \( \text{New} = \{\beta\} \cup \text{supp}(G_1) \cup \text{supp}(G_2) \cup \text{supp}(G_3) \cup \text{New}_1 \cup \text{New}_2, \) where \( \beta \) is the new type variable used at (G.6), \( G_1, G_2 \) and \( G_3 \) are respectively the substitutions for \( \theta_1 \geq \beta \rightarrow \rho \) at (G.6), \( \theta_2 \geq S_1(\beta \rightarrow \rho) \) at (G.7), and \( \theta_3 \geq S_2S_1\beta \) at (G.8), and \( \text{New}_1 \) and \( \text{New}_2 \) are respectively the sets of the new type variables used by \( G(\Gamma, e_1, \theta_1) \) at (G.6) and \( G(S_2S_1\Gamma, e_2, \theta_3) \) at (G.8).

By the (APP) rule, there exists a type \( \tau \) such that
\[
PT \vdash e_1 : \tau \rightarrow P\rho \quad (18)
\]

and
\[
PT \vdash e_2 : \tau. \quad (19)
\]
First, we prove $G(\Gamma, e_1, \theta_1)$ at $(G.6)$ succeeds by induction. Let $P' = P_1(\beta) \cup \{\tau / \beta\}$. Then

$$P'G_1\theta_1 = P'(\beta \rightarrow \rho) \quad \text{by the definition of } G_1$$

and $P'G_1\Gamma = P\Gamma$ because $ftv(\Gamma) \cap (supp(G_1) \cup \{\beta\}) = \emptyset$. Hence, applying induction to $G(\Gamma, e_1, \theta_1)$ at $(G.6)$ and (18), there exists a substitution $R_1$ such that

$$(R_1S_1)|_{New_1} = (P'G_1)|_{New_1}. \quad (20)$$

Then $R_1G_2$ unifies $S_1\theta_1$ and $\theta_2$ at $(G.7)$ because, by noting that

$$ftv(S_1\theta_1) \cap supp(G_2)$$

$$\subseteq (itv(S_1) \cup ftv(\theta_1)) \cap supp(G_2) \quad \text{by Lemma 6}$$

$$\subseteq (ftv(\Gamma) \cup New_1 \cup ftv(\theta_1)) \cap supp(G_2) \quad \text{by Lemma 7}$$

$$= \emptyset, \quad (21)$$

$$R_1G_2(S_1\theta_1)$$

$$= R_1S_1\theta_1 \quad \text{by (21)}$$

$$= P'G_1\theta_1 \quad \text{by (20) and because } ftv(\theta_1) \cap New_1 = \emptyset$$

$$= P'(\beta \rightarrow \rho) \quad \text{by the definition of } G_1$$

$$= P'G_1(\beta \rightarrow \rho) \quad \text{because } ftv(\beta \rightarrow \rho) \cap supp(G_1) = \emptyset$$

$$= R_1S_1(\beta \rightarrow \rho) \quad \text{by (20) and because } ftv(\beta \rightarrow \rho) \cap New_1 = \emptyset$$

$$= R_1G_2(\theta_2) \quad \text{by the definition of } G_2.$$

Thus the unification at $(G.7)$ succeeds with $S_2$ such that for a substitution $R_2$, $R_2S_2 = R_1G_2$. Then

$$(R_2S_2S_1)|_{supp(G_2) \cup New_1}$$

$$= (R_1G_2S_1)|_{supp(G_2) \cup New_1} \quad \text{by Lemma 7}$$

$$= (R_1S_1)|_{supp(G_2) \cup New_1} \quad \text{because } supp(G_2) \cap itv(S_1) = \emptyset$$

$$= (P'G_1)|_{supp(G_2) \cup New_1} \quad \text{by (20).} \quad (22)$$

In order to apply induction to $G(S_2S_1\Gamma, e_2, \theta_3)$ at $(G.8)$ and (19), we must prove that there exists a substitution $P_1$ such that $P_1(S_2S_1\Gamma) = P\Gamma$ and $P_1\theta_3 = \tau$. Such $P_1$ is $R_2G_3$. First, note that, by the definition of $G$,

$$supp(G_3) \cap ftv(S_2S_1\Gamma) = \emptyset \quad (23)$$

because

$$ftv(S_2S_1\Gamma)$$

$$\subseteq itv(S_2) \cup itv(S_1) \cup ftv(\Gamma) \quad \text{by Lemma 6}$$

$$\subseteq ftv(\theta_2) \cup ftv(\theta_1) \cup New_1 \cup ftv(\Gamma) \quad \text{by Theorem 2 and Lemma 7}.$$ 

Thus

$$R_2G_3(S_2S_1\Gamma)$$

$$= R_2S_2S_1\Gamma \quad \text{by (23)}$$

$$= P'G_1\Gamma \quad \text{by (22) and because } ftv(\Gamma) \cap (supp(G_2) \cup New_1) = \emptyset$$

$$= P\Gamma \quad \text{because } ftv(\Gamma) \cap (\{\beta\} \cup supp(G_1)) = \emptyset.$$
Second,
\[
R_2 G_3(\theta_1) = R_2 S_2 S_1 \beta \quad \text{by the definition of } G_3
\]
\[
= P' G_1 \beta \quad \text{by (22) and because } \beta \notin \text{supp}(G_2) \cup \text{New}_1
\]
\[
= P' \beta \quad \text{because } \beta \notin \text{supp}(G_1)
\]
\[
= \tau \quad \text{by the definition of } P'.
\]

Thus by induction, \((G, S)\) succeeds with \(S_3\) such that for a substitution \(R_3\),
\[
(R_3 S_3) |_{\text{New}_2} = (R_2 G_3) |_{\text{New}_2}. \tag{24}
\]

Moreover, note that
\[
(R_3 S_3) |_{\text{New}_2 \cup \text{supp}(G_3)} = R_2 |_{\text{New}_2 \cup \text{supp}(G_3)}. \tag{25}
\]

Then \(R_3\) unifies \(S_3 S_2 S_1 \theta_1\) and \(S_3 S_2 S_1 (\beta \rightarrow \rho)\) at \((G, S)\) because
\[
R_3 S_3 S_2 S_1 \theta_1 = R_2 S_2 S_1 \theta_1 \quad \text{by (25) and}
\]
\[
\text{because } \text{fte}(\theta_1) \cap (\text{New}_2 \cup \text{supp}(G_3)) = \emptyset
\]
\[
= P' G_1 \theta_1 \quad \text{by (22) and}
\]
\[
\text{because } \text{fte}(\theta_1) \cap (\text{New}_1 \cup \text{supp}(G_2)) = \emptyset
\]
\[
= P' (\beta \rightarrow \rho) \quad \text{by the definition of } G_1
\]
\[
= P' G_1 (\beta \rightarrow \rho) \quad \text{because } \text{fte}(\beta \rightarrow \rho) \cap \text{supp}(G_1) = \emptyset
\]
\[
= R_2 S_2 S_1 (\beta \rightarrow \rho) \quad \text{by (22) and}
\]
\[
\text{because } \text{fte}(\beta \rightarrow \rho) \cap (\text{New}_1 \cup \text{supp}(G_2)) = \emptyset
\]
\[
= R_3 S_2 S_2 S_1 (\beta \rightarrow \rho) \quad \text{by (25) and}
\]
\[
\text{because } \beta \notin \text{New}_2 \cup \text{supp}(G_3) = \emptyset.
\]

Thus the unification at \((G, S)\) succeeds with \(S_4\) such that for a substitution \(R_4\),
\[
R_4 S_4 = R_3. \tag{26}
\]

Finally, \(R_4\) unifies \(S_3 S_3 \theta_3\) and \(S_4 S_3 S_2 \beta\) at \((G, S)\) because
\[
R_4(S_4 S_3 \theta_3) = R_3 S_3 \theta_3 \quad \text{by (26)}
\]
\[
= R_2 S_2 S_1 \beta \quad \text{by (24) and because } \text{fte}(\theta_3) \cap \text{New}_2 = \emptyset
\]
\[
= R_2 S_2 S_1 \beta \quad \text{by the definition of } G_3
\]
\[
= R_4(S_3 S_2 S_1 \beta) \quad \text{by (25) and (26), and}
\]
\[
\text{because } \beta \notin \text{New}_2 \cup \text{supp}(G_3).
\]

Thus the unification at \((G, S)\) succeeds with \(S_5\) such that for a substitution \(R_5\),
\[
R_5 S_5 = R_4. \tag{27}
\]

Hence \(G(\Gamma, e_1, e_2, \rho)\) succeeds with \(S_5 S_3 S_2 S_2 S_1\).

Now we prove the rest that \((R_5 S_5 S_3 S_2 S_2 S_1) |_{\text{New}} = P |_{\text{New}}\). Note that, by Lemma 6 and 7 and Theorem 2, \(\text{fte}(S_2 S_1) \subseteq \text{fte}(\Gamma) \cup \text{fte}(\theta_1) \cup \text{fte}(\theta_2) \cup \text{New}_1\), hence by the definition of \(G\),
\[
\text{fte}(S_2 S_1) \cap (\text{New}_2 \cup \text{supp}(G_3)) = \emptyset. \tag{28}
\]
Therefore
\[(R_5S_5S_4S_3S_2S_1)|_{New} = (R_4S_4S_3S_2S_1)|_{New} = (R_3S_3S_2S_1)|_{New}\] by (27)
\[(R_2|_{New}) \cup \text{supp}(G_1)|_{New} = (R_2S_2S_1)|_{New}\] by Lemma 8 and (28)
\[(P'G_1)|_{New} = P|_{New}\] because \((\{\beta\} \cup \text{supp}(G_1)) \subseteq \text{New}.

- **Case let** $x = e_1 \text{ in } e_2$: Let the given judgment be $PT \vdash e_1 : P\rho$, and $\text{New} = \{\beta\} \cup \text{supp}(G) \cup \text{New}_1 \cup \text{New}_2$, where $\beta$ is the new type variable introduced at $(G\,11)$, $G$ is the substitution for $\theta \geq S_1 \rho$ at $(G\,12)$, and $\text{New}_1$ and $\text{New}_2$ are respectively the sets of new type variables used by $G(\Gamma, e_1, \beta)$ at $(G\,11)$ and $G(S_1 \Gamma + x : \text{Clos}_{S_1 \Gamma}(S_1 \beta), e_2, \theta)$ at $(G\,12)$.

By the (LET) rule, there exists a type $\tau$ such that
$$PT \vdash e_1 : \tau$$
(29)
and
$$PT + x : \text{Clos}_{PT}(\tau) \vdash e_2 : P\rho.$$ 
(30)

Let $P' = P'|_{\{\beta\} \cup \{\tau/\beta\}}$. Then $P' \beta = \tau$ and $P' \Gamma = PT$ because $\beta \notin \text{ft}(\Gamma)$. Hence by induction, $G(\Gamma, e_1, \beta)$ at $(G\,11)$ and (29) imply that there exists a substitution $R_1$ such that
$$(R_1S_1)|_{New_1} = P'|_{New_1}.$$ 
(31)

Note that
\[R_1G(S_1 \Gamma) = R_1S_1 \Gamma\] because $\text{supp}(G) \cap \text{ft}(S_1 \Gamma) = \emptyset$
by Lemma 6 and 7
\[\vdash PT\] by (31) and because $\text{ft}(\Gamma) \cap \text{New}_1 = \emptyset$
\[PT\] because the new $\beta \notin \text{ft}(\Gamma)$,
and
\[R_1G(\text{Clos}_{S_1 \Gamma}(S_1 \beta))\]
\[\vdash \text{Clos}_{R_1GS_1 \Gamma}(R_1GS_1 \beta)\] by Lemma 4
\[= \text{Clos}_{PT}(R_1S_1 \beta)\] because $\text{supp}(G) \cap \text{ft}(S_1 \beta) = \emptyset$
by Lemma 6 and 7
\[= \text{Clos}_{PT}(P' \beta)\] by (31) and because $\beta \notin \text{New}_1$
\[\vdash \text{Clos}_{PT}(\tau)\] by the definition of $P'$;
that is, $R_1G(S_1 \Gamma + x : \text{Clos}_{S_1 \Gamma}(S_1 \beta)) \vdash PT + x : \text{Clos}_{PT}(\tau)$. Then by Lemma 5 and (30),
$$R_1G(S_1 \Gamma + x : \text{Clos}_{S_1 \Gamma}(S_1 \beta)) \vdash e_2 : P\rho.$$ 
(32)

In order to apply induction to $G(S_1 \Gamma + x : \text{Clos}_{S_1 \Gamma}(S_1 \beta), e_2, \theta)$ at $(G\,12)$ and (32), we have to prove that $R_1G\theta = P\rho$:
\[R_1G(\theta) = R_1S_1 \rho\] by the definition of $G$
\[= P' \rho\] by (31) and because $\text{ft}(\rho) \cap \text{New}_1 = \emptyset$
\[= P\rho\] because the new $\beta \notin \text{ft}(\rho)$.
Thus by induction, $\mathcal{G}(S_1 \Gamma + x: Clos_{S_1 \Gamma}(S_1 \beta), e_2, \theta)$ at (G.12) succeeds with $S_2$ such that for a substitution $R_2$,

$$(R_2 S_2)|_{\text{New}_2} = (R_1 G)|_{\text{New}_2}.$$  \hfill (33)

Moreover, note that

$$(R_2 S_2)|_{\text{supp}(G) \cup \text{New}_2} = R_1|_{\text{supp}(G) \cup \text{New}_2}.$$  \hfill (34)

Then $R_2$ unifies $S_2 \theta$ and $S_2 S_1 \rho$ at (G.13) because

$$R_2(S_2 \theta) = R_1 G \theta \quad \text{by (33) and because ftv(\theta) \cap \text{New}_2 = \emptyset}$$

$$= R_1 S_1 \rho \quad \text{by the definition of } G$$

$$= R_2(S_2 S_1 \rho) \quad \text{by (34) and because, by Lemma 6 and 7, ftv(S_1 \rho) \cap (\text{supp}(G) \cup \text{New}_2) = \emptyset.}$$

Thus the unification at (G.13) succeeds with $S_3$ such that for a substitution $R_3$,

$$R_3 S_3 = R_2.$$  \hfill (35)

Hence, $\mathcal{G}(\Gamma, \text{let } x = e_1 \text{ in } e_2, \rho)$ succeeds with $S_3 S_2 S_1$.

Now we prove the rest that $(R_3 S_3 S_2 S_1)|_{\text{New}} = P|_{\text{New}}$. Note that, by Lemma 7, $\text{itv}(S_1) \subseteq \text{ftv}(\Gamma) \cup \{\beta\} \cup \text{New}_1$, hence by the definition of $\mathcal{G}$,

$$\text{itv}(S_1) \cap (\text{supp}(G) \cup \text{New}_2) = \emptyset.$$  \hfill (36)

Therefore

$$(R_3 S_3 S_2 S_1)|_{\text{New}}$$

$$= (R_2 S_2 S_1)|_{\text{New}}$$

$$= ((R_2 S_2)|_{\text{supp}(G) \cup \text{New}_2} S_1)|_{\text{New}}$$

$$= (R_1|_{\text{supp}(G) \cup \text{New}_2} S_1)|_{\text{New}}$$

$$= (R_1 S_1)|_{\text{New}}$$

$$= P|_{\text{New}}$$

because $\beta \in \text{New}$.

**case fix:** Let the given judgment be $PT \vdash \text{fix } f \lambda x.e : P \rho$ and New = $\text{supp}(G) \cup \text{New'}$ where $G$ is the substitution for $\theta_1 \land \theta_2 \geq \rho$ at (G.14) and New' is the set of new type variables used by $\mathcal{G}(\Gamma + f : \theta_1, \lambda x.e, \theta_2)$ at (G.14).

By the (FIX) rule, $PT + f : P \rho \vdash \lambda x.e : P \rho$. Because $\text{supp}(G) \cap \text{ftv}(\Gamma) = \emptyset$,

$$PGT + f : P \rho \theta_1 \vdash \lambda x.e : P \rho \theta_2.$$  \hfill (G.14)

By induction, $\mathcal{G}(\Gamma + f : \theta_1, \lambda x.e, \theta_2)$ at (G.14) succeeds with $S_1$ such that for a substitution $R_1$,

$$(R_1 S_1)|_{\text{New'}} = (PG)|_{\text{New'}}.$$  \hfill (37)

Then $R_1$ unifies $S_1 \theta_1, S_1 \theta_2$ and $S_1 \rho$ at (G.15) because

$$R_1(S_1 \rho) = PG \rho \quad \text{by (37) and because ftv(\rho) \cap \text{New'} = \emptyset}$$

$$= P \rho \quad \text{because ftv(\rho) \cap \text{supp}(G) = \emptyset}$$

$$= PG(\theta_1 \lor \theta_2) \quad \text{by the definition of } G$$

$$= R_1(S_1 \theta_1 \lor S_1 \theta_2) \quad \text{by (37) and because (ftv(\theta_1) \cup ftv(\theta_2)) \cap \text{New'} = \emptyset.}$$
Thus the unification at (G.15) succeeds with $S_2$ such that for a substitution $R_2$,

$$R_2S_2 = R_1.$$  \hspace{1cm} (38)

Hence $G(\Gamma, \text{fix } f \lambda x.e, \rho)$ succeeds with $S_2S_1$, and $(R_2S_2S_1)|_{\text{New}} = P|_{\text{New}}$ because

$$(R_2S_2S_1)|_{\text{New}} = (R_1S_1)|_{\text{New}} \quad \text{by (38)}$$

$$= (PG)|_{\text{New}} \quad \text{by (37)}$$

$$= P|_{\text{New}} \quad \text{because supp}(G) \subseteq \text{New}. \quad \square$$

4 More Restraining Instance Stops Earlier

The information amount in the type constraints determines how early the algorithm detects type errors. Carrying less informative (restraining) constraints during type-checking sub-expressions makes it more probable that the algorithm successfully infers their types with being less sensitive to the context, hence delays detecting type errors as such.

We say that an instance $A$ of $G$ is more restraining than another instance $A'$ whenever $A$ always passes more restraining constraints than $A'$. The “always” means that the loosening operations preserve the restraining order between the original constraints: for each pair of corresponding loosenings $\theta_i \geq \rho_i$ in $A$ and $\theta'_i \geq \rho'_i$ in $A'$ for the same input, if $\rho_i$ is more restraining than $\rho'_i$ then so is $\theta_i$ than $\theta'_i$.

**Definition 2** ($A \subseteq A'$) Let $A$ and $A'$ be two instances of $G$. $A$ is more restraining than $A'$, written $A \subseteq A'$, if and only if for each pair of corresponding loosenings $\theta_i \geq \rho_i$ during $A(\Gamma,e,\rho)$ and $\theta'_i \geq \rho'_i$ during $A'(\Gamma,e,\rho)$, if $\rho_i = R\rho'_i$ for a substitution $R$ then $\theta_i = (R|_{\text{supp}(P)} \cup P)\theta'_i$ for a substitution $P$ with supp($P$) $\subseteq \text{fix}(\theta'_i) \setminus \text{fix}(\rho'_i)$.

**Lemma 9** $\mathcal{M} \subseteq \mathcal{H} \subseteq \text{OCaml’s} \subseteq \text{SML/NJ’s} \subseteq \mathcal{W}$.

**Proof.** We prove $A \subseteq A'$ for each consecutive pair of the instance algorithms. For each corresponding pair of $\theta \geq \rho$ in algorithm $A$ and $\theta' \geq \rho'$ in algorithm $A'$ with $\rho = R\rho'$ for a substitution $R$, we must find a substitution $P$ such that $\theta = (R|_{\text{supp}(P)} \cup P)\theta'$.

- **case** $A \subseteq \mathcal{H}$: They differ only at (2) (G.6). For $\mathcal{M}$, it is $\beta \rightarrow \rho \geq \beta \rightarrow \rho$. For $\mathcal{H}$, it is $\beta' \rightarrow 2_2 \geq \beta' \rightarrow \rho'$. By the assumption, for a substitution $R$, $R(\beta \rightarrow \rho') = \beta \rightarrow \rho$. Thus $(R|_{\beta_2} \cup \{\rho/\beta_2\})(\beta' \rightarrow 2_2) = R\beta' \rightarrow \rho = \beta \rightarrow \rho$.

- **case** $\mathcal{H} \subseteq \text{OCaml’s}$: They differ only at (2) (G.6). For $\mathcal{H}$, it is $\beta \rightarrow 2_2 \geq \beta \rightarrow 2$. For OCaml’s algorithm, it is $\beta_1' \geq \beta' \rightarrow \rho'$. For any substitution $R$, $(R|_{\beta_1} \cup \{\beta \rightarrow 2_2/\beta_1'\})\beta_1' = \beta \rightarrow 2$.

- **case** $\text{OCaml’s} \subseteq \text{SML/NJ’s}$ :
  - case (1) at (G.3): For OCaml’s, it is $\rho \geq \rho$. For SML/NJ’s, it is $\beta_1' \geq \rho$ or $\rho' \geq \rho'$. By the assumption, $R\rho' = \rho$. Thus $(R|_{\beta_2} \cup \{\rho/\beta_1\})\beta_1' = \rho$ or $(R \cup \{\})\rho' = \rho$.
  - case (2) at (G.6): For OCaml’s, it is $\beta_1 \geq \rho$. For SML/NJ’s, it is $\beta_1' \geq \rho'$. For any substitution $R$, $(R|_{\beta_2} \cup \{\beta_1/\beta_1\})\beta_1' = \beta_1$.
  - case (3) at (G.7): For OCaml’s, it is $S_1(\beta \rightarrow \rho) \geq S_1(\beta \rightarrow \rho)$. For SML/NJ’s, it is $\beta_2' \geq S_1'(\beta' \rightarrow \rho')$. For any substitution $R$, $(R|_{\beta_2} \cup \{S_1(\beta \rightarrow \rho)/\beta_2'\})\beta_2' = S_1(\beta \rightarrow \rho)$.
case (4) at (G.8): For OCaml’s, it is $S_2S_1\beta \geq S_2S_1\beta'$. For SML/NJ’s, it is $\beta'_3 \geq S_2S_1\beta'$. For any substitution $R$, $(R\upharpoonright_{\beta'_3}) \cup \{S_2S_1\beta/\beta'_3\})\beta'_3 = S_2S_1\beta$.

- case (5) at (G.12): For OCaml’s, it is $S_1\rho \geq S_1\rho$. For SML/NJ’s, it is $\beta'_1 \geq S_1\rho'$. For any substitution $R$, $(R\upharpoonright_{\beta'_1}) \cup \{S_1\rho/\beta'_1\})\beta'_1 = S_1\rho$.

- case (6) at (G.14): For OCaml’s, it is $\rho \land \rho \geq \rho$. For SML/NJ’s, it is $\beta'_1 \wedge \beta'_1 \geq \rho'$. For any substitution $R$, $(R\upharpoonright_{\beta'_1}) \cup \{\rho/\beta'_1\})\beta'_1 = \rho$.

• case SML/NJ’s $\subseteq W$:

  - case (1) at (G.3): For SML/NJ’s, it is $\beta_1 \geq \rho$ or $\rho \geq \rho$. For $W$, it is $\beta'_1 \geq \rho'$. For any substitution $R$, $(R\upharpoonright_{\beta'_1}) \cup \{\beta_1/\beta'_1\})\beta'_1 = \beta_1$ or $(R\upharpoonright_{\beta'_1}) \cup \{\rho/\beta'_1\})\beta'_1 = \rho$.

  - case (6) at (G.14): For SML/NJ’s, it is $\beta_1 \wedge \beta_2 \geq \rho$. For $W$, it is $\beta'_1 \wedge \beta_2 \geq \rho'$. For any substitution $R$, $(R\upharpoonright_{\beta'_1,\beta_1}) \cup \{\beta_1/\beta'_1, \beta_1/\beta_2\}) (\beta'_1 \wedge \beta_2) = \beta_1$.

  - other cases: For SML/NJ’s, it is $\beta_i \geq \tau$ for a type $\tau$. For $W$, it is $\beta'_i \geq \tau'$ for a type $\tau'$.

The time of detecting type errors can be formalized by the notion of call string [LY98]. The call string of $G(\Gamma, e, \rho)$ (written $\text{callstring}(G(\Gamma, e, \rho))$) is constructed by starting with the empty call string and appending a tuple $(\Gamma_1, e_1, \rho_1)^d$ (respectively, $(\Gamma_1, e_1, \rho_1)^u$) whenever $G(\Gamma_1, e_1, \rho_1)$ is called (respectively, returned). The $d$ or a superscript indicates the downward or upward movement of the stack pointer when the inference algorithm is recursively called or returned. Note that the call strings of every instance algorithm of $G$ are always finite, because at most one call to the algorithm occurs for each sub-expression of the program, and that the order of visiting sub-expressions of the input program in every instance algorithm’s call string is the same.

For two instance algorithms $A$ and $A'$ of $G$, if $A$ is more restraining then $A'$ then $A$ stops earlier than $A'$ if the input program is ill-typed:

**Theorem 4** Let $A$ and $A'$ be instances of $G$ such that $A \subseteq A'$, $\Gamma_0$ be a type environment, $e_0$ be an expression, and $\rho_0$ be a type. If $\text{callstring}(A(\Gamma_0, e_0, \rho_0))$ has $(\Gamma_1, e_1, \rho_1)^d/u$, then $\text{callstring}(A'(\Gamma_0, e_0, \rho_0))$ has $(\Gamma'_1, e, \rho')^d/u$ and there exists a substitution $R$ such that $\text{callstring}(R \upharpoonright_{\Gamma}) \succ \Gamma$ and $\text{callstring}(R \upharpoonright_{\rho}) = \rho$.

Because the order of visiting sub-expressions during the execution of the two instance algorithms are the same, the above theorem implies that if $A$ is more restraining than $A'$ then the length (the number of tuples $\text{callstring}(A(\Gamma_0, e_0, \rho_0))$ of $A$’s call string is shorter than or equal to that $\text{callstring}(A'(\Gamma_0, e_0, \rho_0))$ of $A'$’s call string, i.e., $A$ stops earlier than $A'$.

The proof of Theorem 4 uses Lemmas 10 and 11.

**Lemma 10** [LY98] If $\Gamma \succ \Gamma'$ then $\text{callstring}(\Gamma) \succ \text{callstring}(\Gamma')$.

**Lemma 11** Let $A$ and $A'$ be instances of $G$, $\Gamma$ and $\Gamma'$ be type environments, and $\rho$ and $\rho'$ be types such that $\text{callstring}(\Gamma) \succ \Gamma$ and $\text{callstring}(\rho) = \rho$ for a substitution $R$. If $A(\Gamma, e, \rho)$ succeeds with $S$, then $A'(\Gamma', e, \rho')$ succeeds with $S'$ and there exists a substitution $R'$ such that $(R'S')|_{\text{New}} = (SR)|_{\text{New}}$ where New is the set of new type variables used by $A'(\Gamma', e, \rho')$.

**Proof.** Because $A(\Gamma, e, \rho)$ succeeds with $S$, by the soundness of $A$,

$$ST \vdash e : Sp.$$

By Lemma 2, $SRT' \succ ST$ and $S\rho = SR\rho'$. Thus by Lemma 5,

$$SRT' \vdash e : SR\rho'.$$
By the completeness of $A'$, $A'(\Gamma', e, \rho')$ succeeds with $S'$ and there exists a substitution $R'$ such that
\[(R'S')|_{new} = (SR)|_{new}. \]

**Proof of Theorem 4.** We prove by induction on the length of the prefixes of $[A(\Gamma_0, e_0, \rho_0)]$. We add superscript prime (') to all names used by $A'(\Gamma_0, e_0, \rho_0)$.

- **base case:** When the prefixes are of length 1, they represent the initial calls where $e$ is $e_0$. Then the identity substitution $R$ satisfies $R\Gamma_0 \succ \Gamma_0$ and $R\rho_0 = \rho_0$.

Followings are inductive cases. We first prove for the case that the string ends with a return:

$$(\Gamma_0, e_0, \rho_0)^d \cdots (\Gamma, e, \rho)^n.$$

- **case of the return from $c$:** The case means that $[A(\Gamma_0, e_0, \rho_0)]$ has

$$(\Gamma, e, \rho)^d \cdots (\Gamma, e, \rho)^n.$$

By induction hypothesis, $[A'(\Gamma_0, e_0, \rho_0)]$ has $(\Gamma', e, \rho')^d$ and there exists a substitution $R$ such that $R\rho' = \rho$ and $R\Gamma' \succ \Gamma$. Then by Lemma 11, $A'(\Gamma', e, \rho')$ succeeds; that is, $[A'(\Gamma_0, e_0, \rho_0)]$ has $(\Gamma', e, \rho')^n$.

Now we prove the cases that the string ends with a call: $(\Gamma_0, e_0, \rho_0)^d \cdots (\Gamma, e, \rho)^d$.

- **case $e$ in $\lambda x.e$:** that is, $[A(\Gamma_0, e_0, \rho_0)]$ has

$$(\Gamma, \lambda x.e, \rho)^d (S_1 \Gamma + x:S_1\beta_1, e, S_1\beta_2)^d$$

where $S_1 = U(\beta_1 \rightarrow \beta_2, \theta)$ at (G.3), and $\beta_1$ and $\beta_2$ are the new type variables at (G.3).

By induction, $[A'(\Gamma_0, e_0, \rho_0)]$ has $(\Gamma', \lambda x.e, \rho')^d$ and there exists a substitution $R$ such that

$$R\Gamma' \succ \Gamma$$

and $R\rho' = \rho$.

In order for $A'(\Gamma', \lambda x.e, \rho')$ to have a call for $e$, the unification at (G.3) must hold. Because $A \subseteq A'$, there exists a substitution $P$ such that

$$\theta = (R|_{supp(P)} \cup P)\theta'$$

and $supp(P) \subseteq ftv(\theta') \setminus ftv(\rho')$. Note that by the definition of $G$,

$$supp(P) \cap ftv(\Gamma') = \emptyset.$$  \hspace{1cm} (41)

Let $R_0 = R|_{\{\beta_1, \beta_2\} \cup supp(P)} \cup P \cup \{\beta_1/\beta'_1, \beta_2/\beta'_2\}$ where $\beta'_1$ and $\beta'_2$ are the new type variables of $A'$ introduced at (G.3). Then $S_1R_0$ unifies $\beta'_1 \rightarrow \beta'_2$ and $\theta'$ at (G.3) because

$$S_1R_0(\theta') = S_1(R|_{supp(P)} \cup P)\theta' \quad \text{because the new } \beta'_1, \beta'_2 \not\in ftv(\theta')$$

$$= S_1\theta \hspace{1cm} \text{by (40)}$$

$$= S_1(\beta_1 \rightarrow \beta_2) \hspace{1cm} \text{by (G.3)}$$

$$= S_1R_0(\beta'_1 \rightarrow \beta'_2) \hspace{1cm} \text{by the definition of } R_0.$$  \hspace{1cm} (40)

Thus the unification of $A'$ at (G.3) succeeds with $S_1^\theta$, hence $[A'(\Gamma_0, e_0, \rho_0)]$ has $(S_1^\theta \Gamma' + x:S_1^\theta\beta_1, e, S_1^\theta\beta_2)^d$. 
Now we prove the rest that there exists a substitution $R'$ such that $R'(S_1[^\Gamma + x:S_1[^\beta_1]_1]) = (S_1[^\Gamma + x:S_1[^\beta_1]_1])$ and $R'(S_1[^\beta_2]) = S_1[^\beta_2]$. Because $(G.3)$ succeeds with $S_1',$ by Theorem 2, there exists a substitution $R_1$ such that
\[
S_1R_0 = R_1S_1'.
\] (42)

Then such $R'$ is $R_1$ because
\[
R_1(S_1[^\Gamma + x:S_1[^\beta_1]_1]) = S_1R_0(^\Gamma' + x:^\beta_1') \quad \text{by (42)}
\]
\[
= S_1((R_{\supp(P)} \cup P)^\Gamma' + x:R_0[^\beta_1]) \quad \text{because the new } ^\beta_1', ^\beta_2 \notin \text{ftv}(^\Gamma')
\]
\[
= S_1(R^\Gamma' + x:^\beta_1) \quad \text{by (41) and the definition of } R_0
\]
\[
\supset S_1(\Gamma + x:^\beta_1) \quad \text{by (39) and Lemma 2}
\]

and
\[
R_1(S_1[^\beta_2]) = S_1R_0[^\beta_2] \quad \text{by (42)}
\]
\[
= S_1[^\beta_2] \quad \text{by the definition of } R_0.
\]

**case e in e;** that is, $[A(\Gamma_0, e_0, \rho_0)]$ has
\[
(\Gamma, e, e_2, \rho)^d(\Gamma, e, \theta_1)^d
\]
where $\theta_1$ is the type loosened from $\beta \rightarrow \rho$ at $(G.8)$. By induction hypothesis, $[A'(\Gamma_0, e_0, \rho_0)]$ has $(\Gamma', e, e_2, \rho')^d$ and there exists a substitution $R$ such that
\[
R^\Gamma' \supset \Gamma
\] (43)
and $R\rho' = \rho$. Thus by the definition of $G$, $[A'(\Gamma_0, e_0, \rho_0)]$ has $(\Gamma', e, \theta_1')^d$ where $\theta_1'$ is the type loosened from $\beta' \rightarrow \rho'$ at $(G.8)$.

Now we prove the rest. Let $R_0 = R_{\supp(P)} \cup \{\beta/\beta'\}$ where $\beta$ and $\beta'$ are respectively the new type variables of $A$ and $A'$ at $(G.6)$. Because $A \subseteq A'$ and
\[
R_0(\beta \rightarrow \rho') = \beta \rightarrow R\rho' \quad \text{because the new } \beta' \notin \text{ftv}(\rho')
\]
\[
= \beta \rightarrow \rho, \quad \text{there exists a substitution } P \text{ such that}
\]
\[
(R_0)^\supp(P) \cup P)^\theta_1' = \theta_1
\]
and $\supp(P) \subseteq \text{ftv}(\theta_1') \setminus \text{ftv}(\beta' \rightarrow \rho')$. Note that $\supp(P) \cap \text{ftv}(\Gamma') = \emptyset$ by the definition of $G$. Thus
\[
(R_0)^\supp(P) \cup P)^\Gamma' = R^\Gamma' \quad \text{because } \{\beta \cup \supp(P)\} \cap \text{ftv}(\Gamma') = \emptyset
\]
\[
\supset \Gamma \quad \text{by (43)}.
\]

**case e in e;** that is, $[A(\Gamma_0, e_0, \rho_0)]$ has
\[
(\Gamma, e_1, e, \rho)^d(\Gamma, e_1, \theta_1)^d \cdots (\Gamma, e_1, \theta_3)^d(S_2S_1 \Gamma, e, \theta_3)^d
\]
where $\theta_1$, $\theta_2$, and $\theta_3$ are respectively the loosened types of $A$ at $(G.6)$, $(G.7)$, and $(G.8)$, $S_1 = G(\Gamma, e_1, \theta_1)$ at $(G.6)$, and $S_2 = U(S_1 \theta_1, \theta_2)$ at $(G.7)$.
By induction hypothesis, \([A'(\Gamma_0, e_0, \rho_0)]\) has \((\Gamma', e_1, \epsilon, \rho')d\) and there exists a substitution \(R\) such that

\[R\Gamma' \Rightarrow \Gamma \tag{44}\]

and \(R\rho' = \rho\).

In order for \(A'(\Gamma', e_1, \epsilon, \rho')\) to have a call for \(e_1\), its call for \(e_1\) at \((G.6)\) must return and the unification at \((G.7)\) must succeed.

- \(A'(\Gamma', e_1, \theta'_1)\) at \((G.6)\) returns: Let \(R_0 = R_0|_{\{\beta'/\beta\} \cup \{\beta/\beta'\}}\) where \(\beta\) and \(\beta'\) are the new type variables of \(A\) and \(A'\), respectively, introduced at \((G.6)\). Because \(A \sqsubseteq A'\) and

\[R_0(\beta' \rightarrow \rho') = \beta \rightarrow R\rho'\] because the new \(\beta' \notin ftv(\rho')\]

there exists a substitution \(P_1\) such that

\[\theta_1 = (R_0|_{supp(P_1) \cup P_1})\theta'_1 \tag{46}\]

and \(supp(P_1) \subseteq ftv(\theta'_1) \setminus ftv(\beta' \rightarrow \rho')\). Note that by the definition of \(G\),

\[supp(P_1) \cap (ftv(\Gamma') \cup ftv(\beta' \rightarrow \rho')) = \emptyset \tag{47}\]

and thus

\[(R_0|_{supp(P_1) \cup P_1})\Gamma' = R\Gamma' \text{ by (47) and because } \beta' \notin ftv(\Gamma') > \Gamma \text{ by (44)}. \tag{48}\]

Because \([A(\Gamma_0, e_0, \rho_0)]\) has \((\Gamma, e_1, \theta_1)^*, (R_0|_{supp(P_1) \cup P_1})\Gamma' \Rightarrow \Gamma (48)\), and \((R_0|_{supp(P_2) \cup P_2})\theta'_1 = \theta_1 (46)\), by Lemma 11, \(A'(\Gamma', e_1, \theta'_1)\) succeeds with \(S'_1\) such that for a substitution \(R_1\),

\[(R_1 S'_1)|_{New_1} = (S_1(R_0|_{supp(P_1) \cup P_1})|_{New_1}) \tag{49}\]

where \(New_1\) is the set of new type variables used by \(A'(\Gamma', e_1, \theta'_1)\).

- \(U(S'_1, \theta'_2)\) at \((G.7)\) succeeds: Because \(A \sqsubseteq A'\) and

\[R_1(S'_1(\beta' \rightarrow \rho')) = S_1(R_0|_{supp(P_1) \cup P_1})(\beta' \rightarrow \rho')\]

by (49) and because \(ftv(\beta' \rightarrow \rho') \cap New_1 = \emptyset\)

\[= S_1 R_0(\beta' \rightarrow \rho') \text{ by (47)}\]

\[= S_1(\beta \rightarrow \rho) \text{ by (45)}. \tag{50}\]

there exists a substitution \(P_2\) such that

\[\theta_2 = (R_1|_{supp(P_2) \cup P_2})\theta'_2 \tag{51}\]

and \(supp(P_2) \subseteq ftv(\theta'_2) \setminus ftv(S'_1(\beta' \rightarrow \rho'))\). Note that

\[ftv(S'_1(\epsilon' \rightarrow \epsilon')) \cup ftv(S'_1(\beta') \cup ftv(S'_1 \Gamma')) \subseteq ftv(S'_1(\epsilon' \rightarrow \epsilon')) \cup ftv(S'_1(\beta') \cup ftv(S'_1 \Gamma')) \text{ by Lemma 6}\]

\[\subseteq New_1 \cup ftv(\theta'_1) \cup \{\beta'\} \cup ftv(\Gamma') \text{ by Lemma 7}\]
and thus by the definition of $\mathcal{G}$,

$$supp(P_2) \cap (fte(S'_1 \theta'_1) \cup fte(S'_2 \beta') \cup fte(S'_1 \Gamma')) = \emptyset. \quad (52)$$

Then $S_2(R_1 |_{supp(P_2)} \cup P_2)$ unifies $S'_1 \theta'_1$ and $\theta'_2$ at $(\mathcal{G},7)$ because

$$S_2(R_1 |_{supp(P_2)} \cup P_2)(S'_1 \theta'_1) = S_2R_1S'_1 \theta'_1 \quad \text{by (52)}$$

$$= S_2S_1(R_0 |_{supp(P_1)} \cup P_1) \theta'_1 \quad \text{by (54) and because fte(\theta'_1) \cap New_1 = \emptyset}$$

$$= S_2S_1 \theta_1 \quad \text{by (46)}$$

$$= S_2 \theta_2 \quad \text{by (52), \quad (53)}$$

$$= S_2(R_1 |_{supp(P_2)} \cup P_2)(\theta'_2) \quad \text{by (51).}$$

Thus the unification of $A'$ at $(\mathcal{G},7)$ succeeds with $S'_2$.

Therefore $[A'(\Gamma_0,e_0,\rho_0)]$ has $(S'_2S'_1, e, \theta'_2)^d$.

Now we prove the rest that there exists a substitution $R'$ such that $R'_\theta'_2 = \theta_3$ and $R'(S'_2S'_1 \Gamma') \triangleright S_2S_1 \Gamma$. Because $(\mathcal{G},7)$ succeeds with $S'_2$, by Theorem 2, there exists a substitution $R_2$ such that

$$R_2S'_2 = S_2(R_1 |_{supp(P_2)} \cup P_2). \quad (53)$$

Because $A \subseteq A'$ and

$$R_2(S'_2S'_1 \beta') = S_2(R_1 |_{supp(P_2)} \cup P_2)S'_1 \beta' \quad \text{by (53)}$$

$$= S_2R_1S'_1 \beta' \quad \text{by (52)}$$

$$= S_2S_1 \beta \quad \text{by (50),}$$

there exists a substitution $P_3$ such that

$$\theta_3 = (R_2 |_{supp(P_3)} \cup P_3) \theta'_2$$

and $supp(P_3) \subseteq fte(\theta'_2) \setminus fte(S'_2S'_1 \beta')$. Note again that, by Lemma 6 and 7 and Theorem 2,

$$fte(S'_2S'_1 \Gamma') \subseteq fte(\theta_1) \cup fte(\theta_2) \cup New_1 \cup fte(\Gamma')$$

$$\subseteq supp(P_1) \cup fte(\beta \rightarrow \rho) \cup supp(P_2) \cup New_1 \cup fte(\Gamma')$$

and thus by the definition of $\mathcal{G}$,

$$supp(P_3) \cap fte(S'_2S'_1 \Gamma') = \emptyset. \quad (54)$$

Therefore, such $R'$ is $(R_2 |_{supp(P_3)} \cup P_3)$ because

$$(R_2 |_{supp(P_3)} \cup P_3)(S'_2S'_1 \Gamma') = R_2S'_2S'_1 \Gamma' \quad \text{by (54)}$$

$$= S_2(R_1 |_{supp(P_3)} \cup P_2)S'_1 \Gamma' \quad \text{by (53)}$$

$$= S_2R_1S'_1 \Gamma' \quad \text{by (52)}$$

$$= S_2S_1(R_0 |_{supp(P_1)} \cup P_1) \Gamma' \quad \text{by (49) and because fte(\Gamma') \cap New_1 = \emptyset}$$

$$\triangleright S_2S_1 \Gamma \quad \text{by (48) and Lemma 2.}$$
• case e in (let x = e in e2); that is, \([A(\Gamma_0, e_0, \rho_0)]\) has 
\[(\Gamma, \text{let } x = e \text{ in } e_2, \rho)^d(\Gamma, e, \beta)^d\]
where \(\beta\) is the new type variable introduced at \((\mathcal{G}, 11)\). By induction, \([A'(\Gamma_0, e_0, \rho_0)]\) has \((\Gamma', \text{let } x = e \text{ in } e_2, \rho')^d\) and there exists a substitution \(R\) such that \(RI' > \Gamma\) and \(R\rho' = \rho\). By the definition of \(\mathcal{G}\), \([A'(\Gamma_0, e_0, \rho_0)]\) has \((\Gamma', e_1, \beta')^d\) where \(\beta'\) is the new type variable introduced at \((\mathcal{G}, 11)\). Let \(R_0 = R|_{\{\beta'\} \cup \{\beta/\beta'\}}\). Then \(R_0\Gamma' = RI' > \Gamma\) and \(R_0\beta' = \beta\).

• case e in (let x = e1 in e); that is, \([A(\Gamma_0, e_0, \rho_0)]\) has 
\[(\Gamma, \text{let } x = e_1 \text{ in } e, \rho)^d(\Gamma, e, \beta)^d \cdots (\Gamma, e, \beta)^u(S_1\Gamma + x: \text{Clos}_{S_1\Gamma}(S_1\beta), e, \theta)^d\]
where \(\beta\) is the new type variable introduced at \((\mathcal{G}, 11)\), \(\theta\) is the loosened type at \((\mathcal{G}, 12)\), and \(S_1 = G(\Gamma, e_1, \beta)\) at \((\mathcal{G}, 11)\). By induction, \([A'(\Gamma_0, e_0, \rho_0)]\) has \((\Gamma', \text{let } x = e \text{ in } e_2, \rho')^d\) and there exists a substitution \(R\) such that 
\[RI' > \Gamma\]
and \(R\rho' = \rho\). Let \(R_0 = R|_{\{\beta'\} \cup \{\beta/\beta'\}}\) where \(\beta'\) is the new type variable introduced at \((\mathcal{G}, 11)\). Then 
\[R_0\Gamma' = RI' \text{ because the new } \beta' \notin \text{ftv}(\Gamma') \]
\[\geq \Gamma \quad \text{by (55)}\]
and \(R_0\beta' = \beta\). Thus by Lemma 11, \(A'(\Gamma', e_1, \beta')\) at \((\mathcal{G}, 11)\) succeeds with \(S_1'\), hence \([A'(\Gamma_0, e_0, \rho_0)]\) has \((S_1'\Gamma' + x: \text{Clos}_{S_1'\Gamma'}(S_1'\beta'), e_2, \theta')^d\).

Now we prove the rest that there exists a substitution \(R'\) such that \(R'\theta' = \theta\) and \(R'(S_1'\Gamma' + x: \text{Clos}_{S_1'\Gamma'}(S_1'\beta')) \succ S_1\Gamma + x: \text{Clos}_{S_1\Gamma}(S_1\beta)\). Because \((\mathcal{G}, 11)\) succeeds with \(S_1'\), by Lemma 11, there is a substitution \(R_1\) such that 
\[(R_1S_1|_{\text{New}_1} = (S_1R_0)|_{\text{New}_1}\]
(57)
where \(\text{New}_1\) is the set of new type variables used by \(A'(\Gamma', e_1, \beta')\). Because \(A \subseteq A'\) and 
\[R_1(S_1'\rho') = S_1R_0\rho' \quad \text{by (57) and because } \text{ftv}(\rho') \cap \text{New}_1 = \emptyset\]
\[= S_1\rho' \quad \text{because the new } \beta' \notin \text{ftv}(\rho')\]
there exists a substitution \(P\) such that 
\[(R_1|_{\text{supp}(P)} \cup P)\theta' = \theta\]
and \(\text{supp}(P) \subseteq \text{ftv}(\theta') \setminus \text{ftv}(S_1'\rho')\). Note that 
\[\text{ftv}(S_1'\Gamma') \cup \text{ftv}(S_1'\beta') \subseteq \text{itv}(S_1') \cup \text{ftv}(\Gamma') \cup \{\beta'\} \subseteq \text{New}_1 \cup \text{ftv}(\Gamma') \cup \{\beta'\} \quad \text{by Lemma 6}\]
and thus by the definition of \(\mathcal{G}\), 
\[\text{supp}(P) \cap (\text{ftv}(S_1'\Gamma') \cup \text{ftv}(S_1'\beta')) = \emptyset.\]
Therefore, such $R'$ is $(R_1|_{\text{supp}(P)} \cup P)$ because

$$(R_1|_{\text{supp}(P)} \cup P)(S'_{1\Gamma'})$$

$$= R_1(S'_{1\Gamma'}) \quad \text{by (58)}$$

$$= S_1 R_0 \Gamma' \quad \text{by (57) and because } New_1 \cap \text{fte}(\Gamma') = \emptyset$$

$$\succ S_1 \Gamma \quad \text{by (56) and Lemma 2}$$

(59)

and

$$(R_1|_{\text{supp}(P)} \cup P)(\text{Clos}_{S'_{1\Gamma'}}(S_{1\beta'}'))$$

$$= R_1 \text{Clos}_{S'_{1\Gamma'}}(S_{1\beta'}') \quad \text{by (58)}$$

$$\succ \text{Clos}_{S_1 \Gamma'}(R_1 S_{1\beta'}') \quad \text{by Lemma 4}$$

$$\succ \text{Clos}_{S_1 \Gamma}(S_1 R_0 \beta') \quad \text{by (59) and because } \beta' \not\in New_1$$

$$\succ \text{Clos}_{S_1 \Gamma}(S_1 \beta) \quad \text{by the definition of } R_0.$$

- **case** $e$ in $(\text{fix } \lambda x.e)$; that is, $[A(\Gamma_0, e_0, \rho_0)]$ has

$$(\Gamma, \text{fix } \lambda x.e, \rho)^d (\Gamma + f; \theta_1, \lambda x.e, \theta_2)^d$$

where $\theta_1$ and $\theta_2$ are the loosened types at $(G,14)$. By induction, $[A'(\Gamma_0, e_0, \rho_0)]$ has

$$(\Gamma', \text{fix } \lambda x.e, \rho')^d$$

and there exists a substitution $P$ such that

$$R' \Gamma' \succ \Gamma$$

(60)

and $R' \rho' = \rho$. By the definition of $G$, $[A'(\Gamma_0, e_0, \rho_0)]$ has $(\Gamma' + f; \theta_1', \lambda x.e, \theta_2')^d$.

Now we prove the rest. Because $A \sqsupseteq A'$ and $R' \rho' = \rho$, there exists a substitution $P$ such that

$$\theta_1 = (R_1|_{\text{supp}(P)} \cup P)\theta_1'$$

(61)

and

$$\theta_2 = (R_1|_{\text{supp}(P)} \cup P)\theta_2'.$$

Therefore

$$(R_1|_{\text{supp}(P)} \cup P)(\Gamma' + f; \theta_1')$$

$$= R' \Gamma' + f; \theta_1' \quad \text{by (61) and because } \text{fte}(\Gamma') \cap \text{supp}(P) = \emptyset$$

$$\succ \Gamma + f; \theta_1 \quad \text{by (60).}$$

By Lemma 9 and Theorem 4, the following order holds:

**Corollary 1** Let $\Gamma$ be a type environment, $e$ be an expression and $\rho$ be a type.

$$[M(\Gamma, e, \rho)] \leq [H(\Gamma, e, \rho)] \leq [OCaml's(\Gamma, e, \rho)] \leq [SML/NJ's(\Gamma, e, \rho)] \leq [W(\Gamma, e, \rho)]$$

where $|s|$ is the number of tuples in call string $s$. 
5 Conclusion

The two opposite algorithms $W$ and $M$ for the the Hindley/Milner let-polymorphic type system are two extremes in type-checking. The de facto standard Algorithm $W$ is context-insensitive, finding type errors too late, while top-down folklore algorithm $M$ finds type errors too early, by being as much context-sensitive as possible. In realistic compiler systems we need algorithms that avoid this extreme behaviors of the two algorithms, but there exists no systematic way to design such hybrid algorithms.

We presented a generalized let-polymorphic type inference algorithm, from which, by changing its degree of context-sensitivity, various hybrid algorithms can be instantiated. We proved that any of $G$’s instances is sound and complete with respect to the Hindley/Milner let-polymorphic type system, and showed a condition on two instance algorithms so that one algorithm should find type errors earlier than the other. Thus, every instance algorithm’s soundness, completeness, and relative earliness in detecting type errors follow automatically. The set of instances of $G$ includes the two opposite algorithms ($W$ and $M$) and is a superset of those hybrid algorithms used in the SML/NJ\cite{sml99} and OCaml\cite{LRVD99}.

There exists at least one more way to further generalize $G(\Gamma, e, \rho)$. We can loosen not only the type constraint $\rho$ but also the types in the type environment $\Gamma$ and the substitutions ($S_i$’s from recursive calls). McAdam’s algorithm \cite{McA98} is one extreme in this direction; the substitutions from recursive calls are not accumulated in the type environments for type-checking other sub-expressions. Only after all the recursive calls to sub-expressions return, does it check for any inconsistency then apply the substitutions all at once. It is analogous to $W$ in that it does not take the context into account in type-checking sub-expressions. As we did for $G$, by allowing to loosen the types in the substitutions we can parameterize how much of substitutions are accumulated in the type environments for sub-expressions. This loosening must be made up for by posterior unifications, as in $G$.

In general settings \cite{Hen93, CU96, AW93, Tha94, Rem92} where we view type inference algorithms consist of two separate stages - deriving constraints and solving them - the parameters in our generalized algorithm $G$ can be considered a way to control when to solve the constraints. We delay the constraint-solving by passing loosened constraints to recursive calls, and then solve the delayed constraints by applying posterior unifications.

Appendix

A Proof of Lemma 7

We prove by structural induction on $e$.

- **case $\lambda$**: By Theorem 2, $itv(\mathcal{U}(\rho, i)) \subseteq ftv(\rho) \cup ftv(i) = ftv(\rho)$.

- **case $x$**:

  $itv(\mathcal{U}(\rho, \{\vec{\beta}/\vec{\alpha}\} \tau)) \subseteq ftv(\rho) \cup ftv(\{\vec{\beta}/\vec{\alpha}\} \tau)$ by Theorem 2

  $\subseteq ftv(\rho) \cup ftv(\tau) \setminus \vec{\alpha} \cup \vec{\beta}$

  $= ftv(\rho) \cup ftv(\forall \vec{\alpha}. \tau) \cup \vec{\beta}$

  $= ftv(\rho) \cup ftv(\Gamma(x)) \cup \vec{\beta}$ because $\Gamma(x) = \forall \vec{\alpha}. \tau$

  $\subseteq ftv(\rho) \cup ftv(\Gamma) \cup \vec{\beta}$.

Note that $\vec{\beta}$ is the set of new type variables used by $G(\Gamma, x, \rho)$. 
• case $\lambda x.e$: Let $G$ be the substitution for $\theta \geq \rho$ at (G.3). Note that all the type variables in $\text{supp}(G)$ are new by definition.

\[
\text{itv}(S_1) \subseteq \text{ftv}(\theta) \cup \text{ftv}(\beta_1 \rightarrow \beta_2) \quad \text{by Theorem 2}
\]
\[
\subseteq \text{ftv}(\rho) \cup \text{supp}(G) \cup \{\beta_1, \beta_2\} \quad \text{because } \text{supp}(G) = \text{ftv}(\theta) \setminus \text{ftv}(\rho),
\]
\[
\text{itv}(S_2) \subseteq \text{ftv}(S_1 \Gamma) \cup \text{ftv}(S_1 \beta_1) \cup \text{ftv}(S_1 \beta_2) \cup \text{New}_1 \quad \text{by induction}
\]
\[
\subseteq \text{itv}(S_1) \cup \text{itv}(\Gamma) \cup \{\beta_1, \beta_2\} \cup \text{New}_1 \quad \text{by Lemma 6}
\]

where $\text{New}_1$ is the set of new type variables used by $G(S_1 \Gamma + x:S_1 \beta_1, e, S_1 \beta_2)$ at (G.4), and

\[
\text{itv}(S_3) \subseteq \text{ftv}(S_2 S_1 \theta) \cup \text{ftv}(S_2 S_1 \rho) \quad \text{by Theorem 2}
\]
\[
\subseteq \text{itv}(S_2) \cup \text{itv}(S_1) \cup \text{ftv}(\theta) \cup \text{ftv}(\rho) \quad \text{by Lemma 6}
\]
\[
\subseteq \text{itv}(S_2) \cup \text{itv}(S_1) \cup \text{supp}(G) \cup \text{ftv}(\rho). \]

Therefore $\text{itv}(S_3 S_2 S_1) \subseteq \text{ftv}(\Gamma) \cup \text{ftv}(\rho) \cup (\text{supp}(G) \cup \{\beta_1, \beta_2\} \cup \text{New}_1)$. Note that $\text{supp}(G) \cup \{\beta_1, \beta_2\} \cup \text{New}_1$ is the set of new type variables used by $G(\Gamma, \lambda x.e, \rho)$.

Other cases can be similarly proven. □

References


