New Generation Computing, 22(2004)1-36
Ohmsha, Ltd. and Springer-Verlag

Proofs of a Set of Hybrid Let-Polymorphic Type Inference Algorithms*[1]

Hyunjun EO and Oukseh LEE
Research On Program Analysis System[*2]
Department of Computer Science
Korea Advanced Institute of Science and Technology
373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Korea
Kwangkeun YI
School of Computer Science and Engineering [*3]
Seoul National University
San 56-1, Shilim-dong, Gwanak-gu, Seoul 151-742, Korea
{poisson; cookcu; kwang}@ropas.kaist.ac.kr

Received 30 September 2002
Revised manuscript received 28 November 2002

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 obtain not only the two opposite algorithms (the bottom-up standard algorithm $\mathcal{W}$ and the top-down algorithm $\mathcal{M}$) but also other hybrid algorithms which are used in real compilers. Such instances' soundness and completeness follow automatically, and their relative earliness in detecting type-errors is determined by checking a simple condition. The set of instances of the generalized algorithm is a superset of those used in the two most popular ML compilers: SML/NJ and OCaml.

Keywords: Type System, Type Inference Algorithm, Let-Polymorphism, Generalization.

---

*1 This work is supported by Creative Research Initiatives of the Korean Ministry of Science and Technology
*2 National Creative Research Initiative Center, http://ropas.kaist.ac.kr
*3 Work done while the third author was associated with Korea Advanced Institute of Science and Technology

§1 Introduction

1.1 This Work

In realistic compilers, the let-polymorphic type system[14)]'s two opposite algorithms ($\mathcal{W}$[6,14)] and $\mathcal{M}$[10)]) are not attractive candidates. In order to generate helpful type-error messages we need to balance between their two opposite behaviors in type-checking: the bottom-up algorithm $\mathcal{W}$ is context-insensitive, finding type errors too late, while the top-down algorithm $\mathcal{M}$ is as context-sensitive as possible, finding type errors too early. Because of these behaviors, the Standard ML of New Jersey (SML/NJ[19)]) and Objective Caml (OCaml[11)]) compilers use hybrids of the two algorithms.

Several works[2,3,7,8,13,17,23)] clearly show that other type checking strategies are possible. To systematically explore this space of strategies, as well as to justify the existing hybrid ones, we need a framework (1) for integrating the two opposite algorithms into one algorithm; (2) for assuring that such an integrated algorithm is still sound and complete; and (3) for measuring, if possible, how any two hybrid algorithms differ in behaviour.

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 present a condition on two instance algorithms that ensures that one algorithm always finds type errors earlier than the other. By instantiating the generalized algorithm with different parameters, we can obtain not only the two opposite algorithms ($\mathcal{W}$ and $\mathcal{M}$) but also other hybrid algorithms that lie within this spectrum. The set of hybrid algorithms captured by the generalized algorithm is a superset of the existing hybrid algorithms in SML/NJ and OCaml. Within this algorithmic framework, compiler developers can freely experiment with various combinations without the burden of proving their correctness every time.

1.2 Notation

We use the same conventional notation as used in Lee and Yi's.[10)] 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 $\mathit{ftv}(\sigma)$ of free type variables in σ is $\mathit{ftv}(\tau) \setminus \vec{\alpha}$, where $\mathit{ftv}(\tau)$ is the set of type variables in type τ. For a type environment Γ, $\mathit{ftv}(\Gamma) = \cup_{x \in \mathit{dom}(\Gamma)} \mathit{ftv}(\Gamma(x))$. A (simultaneous) substitution $S = \{\tau_i/\alpha_i \mid 1 \leq i \leq n\}$ substitutes type τ_i for type variable α_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 $S\vec{\alpha}$ for $\{S\alpha_1, \cdots, S\alpha_n\}$. For a substitution S, the support $\mathit{supp}(S)$ is $\{\alpha \mid S\alpha \neq \alpha\}$, and the set $\mathit{itv}(S)$ of involved type variables is $\{\alpha \mid \beta \in \mathit{supp}(S), \alpha \in \{\beta\} \cup \mathit{ftv}(S\beta)\}$. For a substitution S and a type τ, $S\tau$ is the type resulting from applying every substitution component τ_i/α_i in S to τ. Hence, $\{\}\tau = \tau$. For a substitution S and a type scheme $\sigma = \forall\vec{\alpha}.\tau$, $S\sigma = \forall\vec{\beta}.S\{\vec{\beta}/\vec{\alpha}\}\tau$, where $\vec{\beta} \cap (\mathit{itv}(S) \cup \mathit{ftv}(\sigma)) = \emptyset$. For a substitution S and a type environment Γ, $S\Gamma = \{x \mapsto S\sigma \mid x \mapsto \sigma \in \Gamma\}$. The composition of substitutions S followed by R is written as RS, which is $\{R(S\alpha)/\alpha \mid \alpha \in$

$\mathit{supp}(S)\} \cup \{R\alpha/\alpha \,|\, \alpha \in \mathit{supp}(R) \setminus \mathit{supp}(S)\}$. Two substitutions S and R are equal if and only if $S\alpha = R\alpha$ for every $\alpha \in \mathit{supp}(S) \cup \mathit{supp}(R)$. For a substitution P and a set of type variables V, we write $P\!\restriction_V$ for $\{\tau/\alpha \in P \mid \alpha \notin V\}$. The relation $\forall\vec{\alpha}.\tau' \succ \tau$ holds whenever there exists a substitution S such that $S\tau' = \tau$ and $\mathit{supp}(S) \subseteq \vec{\alpha}$. We write $\Gamma + x\colon \sigma$ to mean $\{y \mapsto \sigma' \,|\, x \neq y, y \mapsto \sigma' \in \Gamma\} \cup \{x \mapsto \sigma\}$. $\mathit{Clos}_\Gamma(\tau)$ is the same as Gen(Γ,τ) in Damas and Milner's,[6] i.e., $\forall\vec{\alpha}.\tau$, where $\vec{\alpha} = \mathit{ftv}(\tau) \setminus \mathit{ftv}(\Gamma)$.

In presenting type-inference algorithms, we use Robinson's unification algorithm:

Theorem 1.1 (Robinson[18])
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 τ and τ', then $\mathcal{U}(\tau, \tau')$ succeeds with S and there exists a substitution R such that $S' = RS$.

Moreover, S involves only variables of τ and τ'.

1.3 Algorithms $\mathcal{W}$ and $\mathcal{M}$

The source language and its Hindley/Milner style let-polymorphic type system are shown in Fig. 1. The two opposite algorithms ($\mathcal{W}$ and $\mathcal{M}$) are shown

Abstract Syntax

Expr	e	$::=$	`()`	constant
		$\mid$	x	variable
		$\mid$	$\lambda x.e$	function
		$\mid$	$e\ e$	application
		$\mid$	`let` x=e `in` e	
		$\mid$	`fix` $f\ \lambda x.e$	
Type	τ	$::=$	ι	constant type
		$\mid$	α	type variable
		$\mid$	$\tau \to \tau$	function type
TypeScheme	σ	$::=$	$\tau \mid \forall\vec{\alpha}.\sigma$	
TypeEnv	Γ	$\in$	*Var* $\to$ *TypeScheme*	type environment

(CON) $$\Gamma \vdash () : \iota$$

(VAR) $$\frac{\Gamma(x) \succ \tau}{\Gamma \vdash x : \tau}$$

(FN) $$\frac{\Gamma + x\colon \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}$$

(APP) $$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}$$

(LET) $$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma + x\colon \mathit{Clos}_\Gamma(\tau_1) \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\texttt{=}e_1\ \texttt{in}\ e_2 : \tau_2}$$

(FIX) $$\frac{\Gamma + f\colon \tau \vdash \lambda x.e : \tau}{\Gamma \vdash \texttt{fix}\ f\ \lambda x.e : \tau}$$

Fig. 1 Language and Its Let-Polymorphic Type System

$$\textit{Subst} \qquad S \subseteq \{\tau/\alpha \mid \alpha \text{ is a type variable, } \tau \text{ is a type}\}$$

$$\mathcal{W}: \textit{TypeEnv} \times \textit{Expr} \rightarrow \textit{Subst} \times \textit{Type}$$

$$\begin{array}{lll}
\mathcal{W}(\Gamma,\ \texttt{()}) & = & (id,\ \iota) \\
\mathcal{W}(\Gamma,\ x) & = & (id,\ \{\vec{\beta}/\vec{\alpha}\}\tau) \ \text{ where } \Gamma(x) = \forall\vec{\alpha}.\tau,\ \text{new } \vec{\beta} \\
\mathcal{W}(\Gamma, \lambda x.e) & = & \text{let } \ (S_1,\ \tau_1) = \mathcal{W}(\Gamma + x\colon \beta,\ e),\ \text{new } \beta \\
 & & \text{in } \ (S_1,\ S_1\beta \rightarrow \tau_1) \\
\mathcal{W}(\Gamma,\ e_1\ e_2) & = & \text{let } \ (S_1,\ \tau_1) = \mathcal{W}(\Gamma,\ e_1) \\
 & & \qquad (S_2,\ \tau_2) = \mathcal{W}(S_1\Gamma,\ e_2) \\
 & & \qquad S_3 = \mathcal{U}(S_2\tau_1,\ \tau_2 \rightarrow \beta),\ \text{new } \beta \\
 & & \text{in } \ (S_3S_2S_1,\ S_3\beta) \\
\mathcal{W}(\Gamma,\ \texttt{let } x\texttt{=}e_1 \texttt{ in } e_2) = & & \\
 & & \text{let } \ (S_1,\ \tau_1) = \mathcal{W}(\Gamma,\ e_1) \\
 & & \qquad (S_2,\ \tau_2) = \mathcal{W}(S_1\Gamma + x\colon \textit{Clos}_{S_1\Gamma}(\tau_1),\ e_2) \\
 & & \text{in } \ (S_2S_1,\ \tau_2) \\
\mathcal{W}(\Gamma,\ \texttt{fix } f\ \lambda x.e) & = & \text{let } \ (S_1,\ \tau_1) = \mathcal{W}(\Gamma + f\colon \beta,\ \lambda x.e),\ \text{new } \beta \\
 & & \qquad S_2 = \mathcal{U}(S_1\beta,\ \tau_1) \\
 & & \text{in } \ (S_2S_1,\ S_2\tau_1)
\end{array}$$

$$\mathcal{M}: \textit{TypeEnv} \times \textit{Expr} \times \textit{Type} \rightarrow \textit{Subst}$$

$$\begin{array}{lll}
\mathcal{M}(\Gamma,\ \texttt{()},\ \rho) & = & \mathcal{U}(\rho,\ \iota) \\
\mathcal{M}(\Gamma,\ x,\ \rho) & = & \mathcal{U}(\rho,\ \{\vec{\beta}/\vec{\alpha}\}\tau) \ \text{where } \Gamma(x) = \forall\vec{\alpha}.\tau,\ \text{new } \vec{\beta} \\
\mathcal{M}(\Gamma,\ \lambda x.e,\ \rho) & = & \text{let } \ S_1 = \mathcal{U}(\rho,\ \beta_1 \rightarrow \beta_2),\ \text{new } \beta_1,\ \beta_2 \\
 & & \qquad S_2 = \mathcal{M}(S_1\Gamma + x\colon S_1\beta_1,\ e,\ S_1\beta_2) \\
 & & \text{in } \ S_2S_1 \\
\mathcal{M}(\Gamma,\ e_1\ e_2,\ \rho) & = & \text{let } \ S_1 = \mathcal{M}(\Gamma,\ e_1,\ \beta \rightarrow \rho),\ \text{new } \beta \\
 & & \qquad S_2 = \mathcal{M}(S_1\Gamma,\ e_2,\ S_1\beta) \\
 & & \text{in } \ S_2S_1 \\
\mathcal{M}(\Gamma, \texttt{let } x\texttt{=}e_1 \texttt{ in } e_2, \rho) = & & \\
 & & \text{let } \ S_1 = \mathcal{M}(\Gamma,\ e_1,\ \beta),\ \text{new } \beta \\
 & & \qquad S_2 = \mathcal{M}(S_1\Gamma + x\colon \textit{Clos}_{S_1\Gamma}(S_1\beta),\ e_2,\ S_1\rho) \\
 & & \text{in } \ S_2S_1 \\
\mathcal{M}(\Gamma,\ \texttt{fix } f\ \lambda x.e,\ \rho) & = & \mathcal{M}(\Gamma + f\colon \rho,\ \lambda x.e,\ \rho)
\end{array}$$

Fig. 2 Algorithm $\mathcal{W}$ and $\mathcal{M}$. Note that every new type variable is distinct from each other, and the set *New* of new type variables introduced at each recursive call to $\mathcal{W}(\Gamma, e)$ (respectively, $\mathcal{M}(\Gamma, e, \rho)$) satisfies $\textit{New} \cap \textit{ftv}(\Gamma) = \emptyset$ (respectively, $\textit{New} \cap (\textit{ftv}(\Gamma) \cup \textit{ftv}(\rho)) = \emptyset$).

in Fig. 2.

Algorithm $\mathcal{W}$ is context-insensitive. It fails only at an application expression. It infers types of two sub-expressions independently and checks later by unification whether those types conflict. Because of this, an erroneous expression is often successfully type-checked (context-insensitively) long before its consequence collides. On the other hand, algorithm $\mathcal{M}$ is as context-sensitive as possible. It carries a type constraint (or an 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 supplied type constraint. For example, for an application expression "$e_1\ e_2$" with a type constraint, say of `int`, the type constraint for e_1 is $\alpha \rightarrow \texttt{int}$ and the constraint for e_2 is the type that α

becomes after the type inference of e_1. For a constant or a variable expression, its type must satisfy the type constraint that the algorithm has supplied to that point.

Example 1.1
To illustrate the difference between $\mathcal{W}$ and $\mathcal{M}$, consider the application expression

```
1 2.
```

$\mathcal{W}$ fails at the application expression *after* having successfully type-checked the two sub-expressions, while $\mathcal{M}$ fails at the left expression `1` because its type `int` conflicts with a function type expected from the context (an application).

§2 The Generalized Algorithm $\mathcal{G}$

2.1 Overview

Our generalized algorithm is based on the top-down, context-sensitive algorithm $\mathcal{M}$. The key observation is that we can vary the type-checking strategy by changing two factors in $\mathcal{M}$: the amount of information in the type constraints and the positions of calls to unification. Algorithm $\mathcal{M}$ carries as much information as possible in its type constraints and applies a unification at every value (constant, variable, and lambda) expression. Algorithm $\mathcal{W}$, on the other hand, carries no information at its type constraints and applies a unification at every application expression. Tuning these two factors yields other type-checking strategies:

Example 2.1
Consider an application expression

```
(IsOne 2):bool
```

where `IsOne` has type `int` $\rightarrow$ `bool`. As we impose less and less constraints in type-checking sub-expressions yet apply more and more checks later, we obtain the following type-checking variations:

- We type-check `IsOne` with constraint $\beta \rightarrow$ `bool`, which is the strongest expectation. After its success, we type-check 2 with the function's domain type `int` as its constraint. ($\mathcal{M}$)
- We type-check `IsOne` with a weaker constraint, $\beta_1 \rightarrow \beta_2$ with β_1 and β_2 being new type variables. The constraint forces `IsOne`'s type to be a function, but does not constrain its domain or range. After its success, we check whether the function's range type is `bool`. Then we type-check `2` with the function's domain type `int` as its constraint.
- We type-check `IsOne` with no constraint. After its success, we check whether the result type is a function type to `bool`. Then we type-check `2` with the function's domain type `int` as its constraint. (OCaml's type inference algorithm)
- We type-check `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 `int` as its constraint. After its success, we check whether the function's range type is `bool`.

- We type-check `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 `int` $\rightarrow$ `bool`.
- We type-check `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 `IsOne`'s type is a function type from `int` to `bool`. ($\mathcal{W}$)

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

Our generalized algorithm is one that allows, wherever possible, the relaxing of the type constraints and yet makes sure that posterior unifications compensate for the relaxation. The places for relaxing the constraints are right before recursive calls for type-checking sub-expressions. The places for posterior unifications that compensate for the relaxed constraints are after the successful returns from the recursive-calls. Some unifications may only partially compensate for the relaxed constraints. Thus, before the original call returns, a final round of unification must be used to enforce any outstanding constraints. For example, consider type-checking the application expression $e_1\, e_2$ with initial constraint ρ. Our algorithm type-checks e_1 with a type constraint that can be more relaxed than the strongest possible constraint $\beta \rightarrow \rho$. Right after its return, it applies a unification that can compensate, not necessarily completely, for the relaxed constraint. It then type-checks the argument expression e_2 with a type constraint that can be more relaxed than the type that β became. After its success, there are no more sub-expressions to type-check, hence it's time to finalize the compensation for the relaxed constraints at the two recursive calls. This is done by two unifications: each one compensates for the relaxed constraint used in type-checking each sub-expression. The unifications check whether the types from the relaxed constraints agree with what the strongest constraint $\beta \rightarrow \rho$ implies.

2.2 Algorithm Definition

The generalized algorithm $\mathcal{G}$ is shown in Fig. 3. As in $\mathcal{M}$, 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 of 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 (7) in the algorithm, the strongest type constraint ρ is relaxed into θ at each recursive call. This relaxed

$$\begin{array}{lll}
\multicolumn{3}{c}{\mathcal{G} : \textit{TypeEnv} \times \textit{Expr} \times \textit{Type} \rightarrow \textit{Subst}} \\
\mathcal{G}(\Gamma, (), \rho) = \mathcal{U}(\rho, \iota) & & (\mathcal{G}.1) \\
\mathcal{G}(\Gamma, x, \rho) = \mathcal{U}(\rho, \{\vec{\beta}/\vec{\alpha}\}\tau), \text{ new } \vec{\beta},\ \Gamma(x) = \forall\vec{\alpha}.\tau & & (\mathcal{G}.2) \\
\mathcal{G}(\Gamma, \lambda x.e, \rho) = & & \\
\quad \text{let} \quad S_1 = \mathcal{U}(\beta_1 \rightarrow \beta_2, \theta), \text{ new } \beta_1, \text{ new } \beta_2, & (1)\ \theta \geq \rho & (\mathcal{G}.3) \\
\quad\quad\quad S_2 = \mathcal{G}(S_1\Gamma + x\colon S_1\beta_1, e, S_1\beta_2) & & (\mathcal{G}.4) \\
\quad\quad\quad S_3 = \mathcal{U}(S_2S_1\theta, S_2S_1\rho) & & (\mathcal{G}.5) \\
\quad \text{in} \quad S_3S_2S_1 & & \\
\mathcal{G}(\Gamma, e_1\ e_2, \rho) = & & \\
\quad \text{let} \quad S_1 = \mathcal{G}(\Gamma, e_1, \theta_1), \text{ new } \beta, & (2)\ \theta_1 \geq \beta \rightarrow \rho & (\mathcal{G}.6) \\
\quad\quad\quad S_2 = \mathcal{U}(S_1\theta_1, \theta_2), & (3)\ \theta_2 \geq S_1(\beta \rightarrow \rho) & (\mathcal{G}.7) \\
\quad\quad\quad S_3 = \mathcal{G}(S_2S_1\Gamma, e_2, \theta_3), & (4)\ \theta_3 \geq S_2S_1\beta & (\mathcal{G}.8) \\
\quad\quad\quad S_4 = \mathcal{U}(S_3S_2S_1\theta_1, S_3S_2S_1(\beta \rightarrow \rho)) & & (\mathcal{G}.9) \\
\quad\quad\quad S_5 = \mathcal{U}(S_4S_3\theta_3, S_4S_3S_2S_1\beta) & & (\mathcal{G}.10) \\
\quad \text{in} \quad S_5S_4S_3S_2S_1 & & \\
\mathcal{G}(\Gamma, \texttt{let}\ x{=}e_1\ \texttt{in}\ e_2, \rho) = & & \\
\quad \text{let} \quad S_1 = \mathcal{G}(\Gamma, e_1, \beta), \text{ new } \beta & & (\mathcal{G}.11) \\
\quad\quad\quad S_2 = \mathcal{G}(S_1\Gamma + x\colon \textit{Clos}_{S_1\Gamma}(S_1\beta), e_2, \theta), & (5)\ \theta \geq S_1\rho & (\mathcal{G}.12) \\
\quad\quad\quad S_3 = \mathcal{U}(S_2\theta, S_2S_1\rho) & & (\mathcal{G}.13) \\
\quad \text{in} \quad S_3S_2S_1 & & \\
\mathcal{G}(\Gamma, \texttt{fix}\ f\ \lambda x.e, \rho) = & & \\
\quad \text{let} \quad \Gamma_1 = \Gamma + f\colon \theta_1, & (6)\ \theta_1 \geq \rho & (\mathcal{G}.14) \\
\quad\quad\quad S_1 = \mathcal{U}(\beta_1 \rightarrow \beta_2, \theta_2), \text{ new } \beta_1, \text{ new } \beta_2, & (7)\ \theta_2 \geq \theta_1 & (\mathcal{G}.15) \\
\quad\quad\quad S_2 = \mathcal{G}(S_1\Gamma_1 + x\colon S_1\beta_1, e, S_1\beta_2) & & (\mathcal{G}.16) \\
\quad\quad\quad S_3 = \mathcal{U}(S_2S_1\theta_1, S_2S_1\theta_2, S_2S_1\rho) & & (\mathcal{G}.17) \\
\quad \text{in} \quad S_3S_2S_1 & &
\end{array}$$

Fig. 3 A Generalized Type Inference Algorithm $\mathcal{G}$. All the type variables in $\mathit{ftv}(\theta) \setminus \mathit{ftv}(\rho)$ for each $\theta \geq \rho$ are new, every new type variable is distinct from each other, and the set *New* of new type variables introduced at each recursive call to $\mathcal{G}(\Gamma, e, \rho)$ satisfies $\mathit{New} \cap (\mathit{ftv}(\Gamma) \cup \mathit{ftv}(\rho)) = \emptyset$.

constraint is one that can be instantiated to ρ by a substitution that ranges over the type variables occurring only in θ (but not ρ):

Definition 2.1 ($\theta \geq \rho$)
Type θ is more general (more relaxed) than type ρ, written $\theta \geq \rho$, if and only if there exists a substitution G such that $G\theta = \rho$ and $\mathit{supp}(G) = \mathit{ftv}(\theta) \setminus \mathit{ftv}(\rho)$.

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

For the lambda expression case $\lambda x.e$ with type constraint ρ, we first decide on the type constraint for the function's body e. It can be any type that is more relaxed than the range type of ρ. We choose such a type by relaxing ρ first, then picking up its range component by unification:

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

Then we use the resulting range type $S_1\beta_2$ as the constraint in type-checking the function's body:

$$S_2 = \mathcal{G}(S_1\Gamma + x\colon S_1\beta_1, e, S_1\beta_2). \tag{$\mathcal{G}.4$}$$

For example, if we choose the θ to be a new type variable, then the unification ($\mathcal{G}$.3) has no effect, hence e's type is inferred without any constraint. The other extreme is to choose θ to be just ρ. Then e's type is inferred with ρ's range type, if ρ is a function type. After returning from the recursive call to e, we have to compensate for passing the relaxed type constraint. This last step is done by checking whether the relaxed constraint θ can agree with the type that its original ρ became:

$$S_3 = \mathcal{U}(S_2S_1\theta, S_2S_1\rho). \qquad (\mathcal{G}.5)$$

Consider type-checking an application expression $e_1\ e_2$ with type constraint ρ. First we decide on the type constraint for the function expression e_1. It can be any type that is more relaxed than the most informative constraint $\beta \to \rho$ with β being a new type variable:

$$S_1 = \mathcal{G}(\Gamma, e_1, \theta_1),\ \text{new } \beta, \quad (2)\ \theta_1 \geq \beta \to \rho. \qquad (\mathcal{G}.6)$$

After the success of this recursive call, we can compensate, not necessarily completely, for passing the relaxed type constraint θ_1. The compensation may be varied according to the constraint we wish to impose on the type of e_1. We can check the result type against the strongest constraint $\beta \to \rho$ or we can check against nothing. Varying the degree of compensation amounts to choosing yet another more relaxed type θ_2 than $S_1(\beta \to \rho)$ and by unifying it with the type that θ_1 became:

$$S_2 = \mathcal{U}(S_1\theta_1, \theta_2), \quad (3)\ \theta_2 \geq S_1(\beta \to \rho). \qquad (\mathcal{G}.7)$$

argument expression e_2. It can be any type that is more relaxed than the type that β became. Hence the next recursive call is:

$$S_3 = \mathcal{G}(S_2S_1\Gamma, e_2, \theta_3), \quad (4)\ \theta_3 \geq S_2S_1\beta. \qquad (\mathcal{G}.8)$$

The finalizing compensation for passing the relaxed type constraints to the two recursive calls is done by checking whether the first relaxed constraint θ_1 can agree with the type that the original type $\beta \to \rho$ became:

$$S_4 = \mathcal{U}(S_3S_2S_1\theta_1, S_3S_2S_1(\beta \to \rho)) \qquad (\mathcal{G}.9)$$

and by checking whether the other relaxed constraint θ_3 for the argument expression can agree with what the original type β became:

$$S_5 = \mathcal{U}(S_4S_3\theta_3, S_4S_3S_2S_1\beta). \qquad (\mathcal{G}.10)$$

We don't have to check for θ_2 because of its unification with θ_1 at line ($\mathcal{G}$.7).

Consider inferring the type of let-expression `let` x=e_1 `in` e_2 with type constraint ρ. Because there is no context information about the type of the first expression e_1, there is no room for varying its type constraint:

$$S_1 = \mathcal{G}(\Gamma, e_1, \beta),\ \text{new } \beta. \qquad (\mathcal{G}.11)$$

Next we decide on the type constraint for the body expression e_2. It can be any type that is more relaxed than the given constraint ρ:

$$S_2 = \mathcal{G}(S_1\Gamma + x\colon \mathit{Clos}_{S_1\Gamma}(S_1\beta), e_2, \theta), \quad (5)\ \theta \geq S_1\rho. \qquad (\mathcal{G}.12)$$

Finally, we have to check whether the relaxed constraint agrees with the type that the original constraint became:

$$\mathcal{G}^R : \mathit{TypeEnv} \times \mathit{Expr} \times \mathit{Type} \rightarrow \mathit{Subst}$$

$$\begin{array}{lll}
\mathcal{G}^R(\Gamma, e_1\ e_2, \rho) = & & \\
\quad \text{let} \quad S_1 = \mathcal{G}^R(\Gamma, e_2, \beta),\ \text{new}\ \beta & & (\mathcal{G}.18) \\
\qquad\quad S_2 = \mathcal{G}^R(S_1\Gamma, e_1, \theta_1), & \theta_1 \geq S_1(\beta \rightarrow \rho) & (\mathcal{G}.19) \\
\qquad\quad S_3 = \mathcal{U}(S_2\theta_1, S_2S_1(\beta \rightarrow \rho)) & & (\mathcal{G}.20) \\
\quad \text{in} \quad S_3S_2S_1 & &
\end{array}$$

Fig. 4 A Generalized Type Inference Algorithm $\mathcal{G}^R$. For $e_1\ e_2$, $\mathcal{G}^R$ infers the type of e_2 first, while $\mathcal{G}$ infers the type of e_1 first. Other parts of $\mathcal{G}^R$ are the same as those of $\mathcal{G}$ except that every recursive call in inference algorithm is $\mathcal{G}^R$, not $\mathcal{G}$.

$$S_3 = \mathcal{U}(S_2\theta, S_2S_1\rho). \qquad (\mathcal{G}.13)$$

The case for recursive function $\texttt{fix}\ f\ \lambda x.e$ is similar. First, we decide on the type constraint for f. It can be any type that is more relaxed that the given constraint ρ:

$$\Gamma_1 = \Gamma + f : \theta_1, \quad (6)\ \theta_1 \geq \rho. \qquad (\mathcal{G}.14)$$

Next we decide on what is expected for the type of $\lambda x.e$. We choose such a type by relaxing θ_1 first, then picking up its domain and range component by unification:

$$S_1 = \mathcal{U}(\beta_1 \rightarrow \beta_2, \theta_2),\ \text{new}\ \beta_1, \beta_2, \quad (7)\ \theta_2 \geq \theta_1. \qquad (\mathcal{G}.15)$$

Then we use the resulting range type $S_1\beta_2$ as the constraint in type-checking the function body and the domain type $S_1\beta_1$ as the type of x:

$$S_2 = \mathcal{G}(S_1\Gamma_1 + x\!: S_1\beta_1, e, S_1\beta_2). \qquad (\mathcal{G}.16)$$

Finally, we check whether the relaxed type constraints agrees with the type that the original constraint became:

$$S_3 = \mathcal{U}(S_2S_1\theta_1, S_2S_1\theta_2, S_2S_1\rho). \qquad (\mathcal{G}.17)$$

We have another variant of generalized type inference algorithm $\mathcal{G}^R$ in Fig. 4. For the function application $e_1\ e_2$, $\mathcal{G}^R$ infers the type of argument expression e_2 first, and then infers the type of function expression e_1. For other expressions, $\mathcal{G}^R$ is the same as $\mathcal{G}$ except that every recursive call in inference algorithm is $\mathcal{G}^R$, not $\mathcal{G}$.

Consider type-checking an application expression $e_1\ e_2$ with type constraint ρ. Because we do not have any context information about the type of the argument e_2, there is no room for varying its type constraint:

$$S_1 = \mathcal{G}^R(\Gamma, e_2, \beta),\ \text{new}\ \beta. \qquad (\mathcal{G}.18)$$

Next we decide on the type constraint for the function expression e_1. It can be any type that is more relaxed than the function type from β to given constraint ρ:

$$S_2 = \mathcal{G}^R(S_1\Gamma, e_1, \theta_1), \quad \theta_1 \geq S_1(\beta \rightarrow \rho). \qquad (\mathcal{G}.19)$$

Finally, we have to check whether the relaxed constraint agrees with the type that the original type $\beta \rightarrow \rho$ became:

$$S_3 = \mathcal{U}(S_2\theta, S_2S_1(\beta \to \rho). \qquad (\mathcal{G}.20)$$

2.3 Instances

By determining the relaxed constraints θ's in $\mathcal{G}$, we obtain various type-inference algorithms, including the standard algorithm $\mathcal{W}$, the top-down algorithm $\mathcal{M}$, and the combinations of the two algorithms used in the SML/NJ[19] and OCaml[11] compiler systems.

- $\mathcal{W}$ is an instance of $\mathcal{G}$ where every θ is a new type variable.
- $\mathcal{M}$ is an instance of $\mathcal{G}$ where every θ is not relaxed: for each case $\theta \geq \rho$ in $\mathcal{G}$, we choose ρ for θ.
- The OCaml's type inference algorithm[*4] is an instance of $\mathcal{G}$ where the θ at (2) (line ($\mathcal{G}$.6)) is a new type variable and other θ's are not relaxed.
- The SML/NJ's type inference algorithm[*5] is an instance of $\mathcal{G}$ where every θ is a new type variable, except that θ_2 at (7) (line ($\mathcal{G}$.15)) is the same with θ_1 at (6) (line ($\mathcal{G}$.14)).
- Other variations than the existing algorithms are also possible from $\mathcal{G}$. For example, consider an instance of $\mathcal{G}$ where the θ at ($\mathcal{G}$.6) is a new function type ($\beta_1 \to \beta_2$ for new variables β_1 and β_2) and other θ's are not relaxed. Let's call this instance algorithm $\mathcal{H}$.

The θ's used in the five instances are summarized in Fig. 5. Please note that for SML/NJ's algorithm, the relaxed constraint for the $\lambda x.e$ case (line $\mathcal{G}$.3) has two candidates, of which we choose one depending on whether the lambda is recursive (defined in `fix` $f\ \lambda x.e$) or not.

	(1) θ	(2) θ_1	(3) θ_2	(4) θ_3	(5) θ	(6) θ_1	(7) θ_2
$\mathcal{W}$	β_3	β_1	β_2	β_3	β_1	β_3	β_4
SML/NJ's	β_3	β_1	β_2	β_3	β_1	β_3	θ_1
OCaml's	ρ	β_1	$S_1(\beta \to \rho)$	$S_2S_1\beta$	$S_1\rho$	ρ	θ_1
$\mathcal{H}$	ρ	$\beta \to \beta_2$	$S_1(\beta \to \rho)$	$S_2S_1\beta$	$S_1\rho$	ρ	θ_1
$\mathcal{M}$	ρ	$\beta \to \rho$	$S_1(\beta \to \rho)$	$S_2S_1\beta$	$S_1\rho$	ρ	θ_1

Fig. 5 Five Instances of Algorithm $\mathcal{G}$. β_i's are new type variables introduced in the θ's.

2.4 Every Instance is Sound and Complete

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

Theorem 2.1 (Soundness)
Let e be an expression, Γ be a type environment, and ρ be a type. If $\mathcal{G}(\Gamma, e, \rho)$ succeeds with S, then $S\Gamma \vdash e : S\rho$. The theorem also holds for $\mathcal{G}^R$.

[*4] We figured out the OCaml's type inference algorithm by examining the source codes of OCaml 3.06.[11]

[*5] We figured out the SML/NJ's type inference algorithm by examining the source codes of SML/NJ 110.0.7.[19]

Proof
See Appendix Section A1. ■

Theorem 2.2 (Completeness)
Let e be an expression, and let Γ be a type environment. If there exist a type ρ and a substitution P such that $P\Gamma \vdash e : P\rho$, then $\mathcal{G}(\Gamma, e, \rho)$ succeeds with S and there exists a substitution R such that $P\!\downharpoonright_{New} = (RS)\!\downharpoonright_{New}$ where *New* is the set of new type variables used by $\mathcal{G}(\Gamma, e, \rho)$. The theorem also holds for $\mathcal{G}^R$.

Proof
See Appendix Section A2. ■

Completeness means that if an expression e has a type τ that satisfies a type constraint ρ (i.e., $\exists P.\tau = P\rho$), then algorithm $\mathcal{G}$ for the expression with the constraint ρ succeeds with substitution S such that the result type $S\rho$ subsumes τ (i.e., the principality, $\exists R.\tau = R(S\rho)$).

2.5 More Restraining Instances of $\mathcal{G}$ Detect Errors Sooner

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 $\mathcal{G}$ is more restraining than another instance A' whenever A always passes more restraining constraints than A'. The "always" means that the relaxing operations preserve the restraining order between the original constraints: for each pair of corresponding relaxations $\theta_i \geq \rho_i$ in A and $\theta'_i \geq \rho'_i$ in A' for the same input, if ρ_i is more restraining than ρ'_i then so is θ_i than θ'_i.

Definition 2.2 ($A \sqsubseteq A'$)
Let A and A' be two instances of $\mathcal{G}$. A is more restraining than A', written $A \sqsubseteq A'$, if and only if for each pair of corresponding relaxations $\theta_i \geq \rho_i$ 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 $\theta_i = (R\!\downharpoonright_{supp(P)} \cup P)\theta'_i$ for a substitution P with $supp(P) \subseteq ftv(\theta'_i) \setminus ftv(\rho'_i)$. We define $A \sqsubseteq A'$ for the instances of $\mathcal{G}^R$ in the same way.

Lemma 2.1
$\mathcal{M} \sqsubseteq \mathcal{H} \sqsubseteq$ OCaml's $\sqsubseteq$ SML/NJ's $\sqsubseteq \mathcal{W}$.

Proof
We prove $A \sqsubseteq A'$ 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 $\theta = (R\!\downharpoonright_{supp(P)} \cup P)\theta'$.

- **case** $\mathcal{M} \sqsubseteq \mathcal{H}$: They differ only at (2) ($\mathcal{G}$.6). For $\mathcal{M}$, it is $\beta \to \rho \geq \beta \to \rho$. For $\mathcal{H}$, it is $\beta' \to \beta'_2 \geq \beta' \to \rho'$. By the assumption, for a substitution R,

$R(\beta' \to \rho') = \beta \to \rho$. Thus $(R\restriction_{\{\beta_2'\}} \cup \{\rho/\beta_2'\})(\beta' \to \beta_2') = R\beta' \to \rho = \beta \to \rho$.

- **case** $\mathcal{H} \sqsubseteq$ OCaml's: They differ only at (2) ($\mathcal{G}$.6). For $\mathcal{H}$, it is $\beta \to \beta_2 \geq \beta \to \rho$. For OCaml's algorithm, it is $\beta_1' \geq \beta' \to \rho'$. For any substitution R, $(R\restriction_{\{\beta_1'\}} \cup \{\beta \to \beta_2/\beta_1'\})\beta_1' = \beta \to \beta_2$.
- **case** OCaml's $\sqsubseteq$ SML/NJ's:
  - case (1) at ($\mathcal{G}$.3): For OCaml's, it is $\rho \geq \rho$. For SML/NJ's, it is $\beta_3' \geq \rho'$. For any substitution R, $(R\restriction_{\{\beta_3'\}} \cup \{\rho/\beta_3'\})\beta_3' = \rho$.
  - case (2) at ($\mathcal{G}$.6): For OCaml's, it is $\beta_1 \geq \rho$. For SML/NJ's, it is $\beta_1' \geq \rho'$. For any substitution R, $(R\restriction_{\{\beta_1'\}} \cup \{\beta_1/\beta_1'\})\beta_1' = \beta_1$.
  - case (3) at ($\mathcal{G}$.7): For OCaml's, it is $S_1(\beta \to \rho) \geq S_1(\beta \to \rho)$. For SML/NJ's, it is $\beta_2' \geq S_1'(\beta' \to \rho')$. For any substitution R, $(R\restriction_{\{\beta_2'\}} \cup \{S_1(\beta \to \rho)/\beta_2'\})\beta_2' = S_1(\beta \to \rho)$.
  - case (4) at ($\mathcal{G}$.8): For OCaml's, it is $S_2S_1\beta \geq S_2S_1\beta$. For SML/NJ's, it is $\beta_3' \geq S_2'S_1'\beta'$. For any substitution R, $(R\restriction_{\{\beta_3'\}} \cup \{S_2S_1\beta/\beta_3'\})\beta_3' = S_2S_1\beta$.
  - case (5) at ($\mathcal{G}$.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, $(R\restriction_{\{\beta_1'\}} \cup \{S_1\rho/\beta_1'\})\beta_1' = S_1\rho$.
  - case (6) at ($\mathcal{G}$.14): For OCaml's, it is $\rho \geq \rho$. For SML/NJ's, it is $\beta_3' \geq \rho'$. For any substitution R, $(R\restriction_{\{\beta_3'\}} \cup \{\rho/\beta_3'\})\beta_3' = \rho$.
  - case (7) at ($\mathcal{G}$.15): OCaml's and SML/NJ's are the same $\theta_1 \geq \theta_1$.
- **case** SML/NJ's $\sqsubseteq \mathcal{W}$:
  - case (7) at ($\mathcal{G}$.15): For SML/NJ's, it is $\theta_1 \geq \theta_1$. For $\mathcal{W}$, it is $\beta_4' \geq \theta_1'$. For any substitution R, $(R\restriction_{\{\beta_4'\}} \cup \{\theta_1/\beta_4'\})\beta_4' = \theta_1$.
  - other cases: For SML/NJ's, it is $\beta_i \geq \tau$ for a type τ. For $\mathcal{W}$, it is $\beta_i' \geq \tau'$ for a type τ'. For any substitution R, $(R\restriction_{\{\beta_i'\}} \cup \{\beta_i/\beta_i'\})\beta_i' = \beta_i$.

■

The time of detecting type errors can be formalized by the notion of *call string*.[10] The call string of $\mathcal{G}(\Gamma, e, \rho)$ (written $[\![\mathcal{G}(\Gamma, e, \rho)]\!]$) 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 $\mathcal{G}(\Gamma_1, e_1, \rho_1)$ is called (respectively, returned). The d or u superscript indicates the *d*ownward or *u*pward movement of the stack pointer when the inference algorithm is recursively called or returned. Note that the call strings of every instance algorithm of $\mathcal{G}$ 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 $\mathcal{G}$, if A is more restraining than A' then A stops earlier than A' if the input program is ill-typed:

Theorem 2.3
Let A and A' be instances of $\mathcal{G}$ such that $A \sqsubseteq A'$, Γ_0 be a type environment, e_0 be an expression, and ρ_0 be a type. If $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma, e, \rho)^{d/u}$, then $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', e, \rho')^{d/u}$ and there exists a substitution R such that $R\Gamma' \succ \Gamma$ and $R\rho' = \rho$. The theorem also holds for $\mathcal{G}^R$.

Proof
See Appendix Section A4. ■

Because the orders 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) $|[\![A(\Gamma_0, e_0, \rho_0)]\!]|$ of A's call string is shorter than or equal to that $|[\![A'(\Gamma_0, e_0, \rho_0)]\!]|$ of A''s call string, i.e., A stops earlier than A'.

By Lemma 2.1 and Theorem 2.3, the following order holds:

Corollary 2.1
Let Γ be a type environment, e be an expression and ρ be a type.

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

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

§3 Discussion

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 $\mathcal{G}$'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. The set of instances of $\mathcal{G}$ includes the two opposite algorithms ($\mathcal{W}$ and $\mathcal{M}$) and is a superset of those hybrid algorithms used in the SML/NJ[19] and OCaml.[11]

Note that the earliness condition cannot be a criterion to judge the algorithm's goodness in detecting the cause of type-errors. For any algorithm there exists an ill-typed program that falsifies its type-error message. The earliness condition can just be a criterion by which compiler developers can achieve different type-checking strategies.

It is possible to further generalize $\mathcal{G}(\Gamma, e, \rho)$. We can relax not only the type constraint ρ but also the type environment Γ. Note that algorithm $\mathcal{G}$ passes the most informative type environment to sub-or-sibling expressions; it accumulates all substitutions in the type environment at its recursive calls. This is a top-down strategy; bottom-up approaches such as Bernstein and Stark's[3] and Chitil's[4] use unconstrained type environments to check sub-or-sibling expressions. Between these two opposing strategies lie hybrid ones.[12,24] These variations can be formalized, similarly to $\mathcal{G}$, by type-environment relaxing and posterior unification.

In general settings[1,5,9,15,16,20~22] where one views type inference algorithms as consisting of two separate stages - deriving constraints and solving them - the parameters in our generalized algorithm $\mathcal{G}$ can be considered a way to control when to solve the constraints within the Hindley/Milner type system. We delay the constraint-solving by passing relaxed constraints to recursive calls, and then solve the delayed constraints by applying posterior unifications.

References

1) Aiken, A. and Wimmers, E. L., "Type Inclusion Constraints and Type Inference," in *Proc. of Functional Programming Languages and Computer Architecture*, pp. 31–41, 1993.

2) Beaven, M. and Stansifer, R., "Explaining Type Errors in Polymorphic Languages," *ACM Letters on Programming Languages and Systems, 2*, pp. 17–30, Mar.-Dec. 1993.

3) Bernstein, K. L. and Stark, E. W., "Debugging Type Errors (Full Version)," *Technical Report*, State University of New York at Stony Brook, 1995.

4) Chitil, O., "Compositional Explanation of Types and Algorithmic Debugging of Type Errors," in *Proc. of the 6th ACM SIGPLAN International Conference on Functional Programming*, pp. 193–204, Sept. 2001.

5) Cho, K. and Ueda, K., "Diagnosing Non-well-moded Concurrent Logic Programs," in *Joint International Conference on Logic Programming*, pp. 215–229, MIT Press, 1996.

6) Damas, L. and Milner, R., "Principal Type-scheme for Functional Programs," in *Proc. of the 9th Annual ACM Symposium on Principles of Programming Languages*, pp. 207–212, ACM Press, New York, 1982.

7) Duggan, D., "Correct Type Explanation," in *Proc. of Workshop on ML*, pp. 49–58, 1998.

8) Duggan, D. and Bent, F., "Explaining Type Inference," *Science of Computer Programming, 27, 1*, pp. 37–83, July 1996.

9) Henglein, F., "Type Inference with Polymorphic Recursion," *ACM Transactions on Programming Languages and Systems, 15, 2*, pp. 253–289, April 1993.

10) Lee, O. and Yi, K., "Proofs about a Folklore Let-polymorphic Type Inference Algorithm," *ACM Transactions on Programming Languages and Systems, 20, 4*, pp. 707–723, July 1998.

11) Leroy, X., Doligez, D., Garrigue, J., Rémy, D. and Vouillon, J., "The Objective Caml System Release 3.06," Institut National de Recherche en Informatique et en Automatique, August 2002. `http://caml.inria.fr`.

12) McAdam, B. J., "On the Unification of Substitutions in Type Inference," in *Proc. of The International Workshop on Implementation of Fuctional Languages* (Hammond, K., Davie, A. J. T. and Clack, C. eds.), *vol. 1595 of Lecture Notes in Computer Science*, pp. 139–154, Springer-Verlag, Sept. 1998.

13) McAdam, B. J., "Generalising Techniques for Type Debugging," in *Proc. of 1st Scottish Functional Programming Workshop*, 1999.

14) Milner, R., "A Theory of Type Polymorphism in Programming," *Journal of Computer and System Sciences, 17*, pp. 348–375, 1978.

15) Odersky, M., Sulzmann, M. and Wehr, M., "Type Inference with Constrained Types," *Theory and Practice of Object Systems, 5, 1*, 1999.

16) Rémy, D., "Extending ML Type System with a Sorted Equational Theory," *Research Report 1766*, Institut National de Recherche en Informatique et Automatisme, Rocquencourt, BP 105, 78 153 Le Chesnay Cedex, France, 1992.

17) Rideau, L. and Théry, L., "Interactive Programming Environment for ML," *Technical Report 3139*, Institut National de Recherche en Informatique et en Automatique, Mar. 1997.

18) Robinson, J. A., "A Machine-oriented Logic Based on the Resolution Principle," *Journal of ACM, 12, 1*, pp. 23–41, Jan. 1965.

19) The Standard ML of New Jersey, release 110.0.7. Bell Labs, Lucent Technologies, Sept. 2001. `http://cm.bell-labs.com/cm/cs/what/smlnj`.

20) Sulzmann, M., "A General Type Inference Framework for Hindley/Milner Style Systems," in *Proc. of 5th International Symposium on Functional and Logic Programming*, Mar. 2001.

21) Sulzmann, M., Müller, M. and Zenger, C., "Hindley/Milner Style Type Systems in Constraint Form," *Technical Report ACRC-99-009*, University of South Australia, School of Computer and Information Science, Jul. 1999.

22) Thatte, S. R., "Type Inference with Partial Types," *Theoretical Computer Science, 124, 1*, pp. 127–148, Feb. 1994.

23) Wand, M., "Finding the Source of Type Errors," in *Proc. of the 13th Annual ACM Symposium on Principles of Programming Languages*, pp. 38–43, ACM Press, New York, 1986.

24) Yang, J., "Explaining Type Errors by Finding the Sources of Type Conflicts," in *Proc. of 1st Scottish Functional Programming Workshop*, pp. 387–401, Aug. 1999.

Appendix A

§A1 Soundness Proof

Theorem 2.1 (Soundness)
Let e be an expression, Γ be a type environment, and ρ be a type. If $\mathcal{G}(\Gamma, e, \rho)$ succeeds with S, then $S\Gamma \vdash e : S\rho$. The theorem also holds for $\mathcal{G}^R$.

The proof uses Lemmas A1.1–A1.3 and Theorem 1.1.

Lemma A1.1 (Damas and Milner[6])
If $\Gamma \vdash e : \tau$, then $S\Gamma \vdash e : S\tau$.

Lemma A1.2 (Damas and Milner[6])
If $\sigma \succ \sigma'$ then $S\sigma \succ S\sigma'$.

Lemma A1.3 (Milner[14])
Let S be a substitution, Γ be a type environment, and τ be a type. $SClos_\Gamma(\tau) = Clos_{S'\Gamma}(S'\tau)$, where $S' = S\{\vec{\beta}/\vec{\alpha}\}$, $\vec{\alpha} = ftv(\tau) \setminus ftv(\Gamma)$ and $\vec{\beta}$ is new.

Proof of Theorem 2.1
We prove by structural induction on e, and we prove for $\mathcal{G}$ and $\mathcal{G}^R$ simultaneously.

- **case** $()$: $S\rho = S\iota = \iota$. So $S\Gamma \vdash ()$: $S\rho$ by (CON).

- **case** x : $S\rho = S\{\vec{\beta}/\vec{\alpha}\}\tau \prec S\Gamma(x)$ by Lemma A1.2. 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 (FN),

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

  By Lemma A1.1, we can apply S_3 to both sides:

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

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

  $$S_3S_2S_1\Gamma \vdash \lambda x.e : S_3S_2S_1\rho.$$

- **case** $e_1\ e_2$ for $\mathcal{G}$: By induction, ($\mathcal{G}$.6) implies $S_1\Gamma \vdash e_1 : S_1\theta_1$. By Lemma A1.1, 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 $S_4S_3S_2S_1\theta_1 = S_4S_3S_2S_1(\beta \to \rho)$ by ($\mathcal{G}$.9) and $S_5S_4S_3S_2S_1\beta = S_5S_4S_3\theta_3$ by ($\mathcal{G}$.10),

  $$S_5S_4S_3S_2S_1\Gamma \vdash e_1 : S_5S_4S_3(\theta_3 \to S_2S_1\rho). \quad (1)$$

  By induction, ($\mathcal{G}$.8) implies $S_3S_2S_1\Gamma \vdash e_2 : S_3\theta_3$. By Lemma A1.1, we can apply S_5S_4 to both sides:

  $$S_5S_4S_3S_2S_1\Gamma \vdash e_2 : S_5S_4S_3\theta_3. \quad (2)$$

  Hence by (APP), (1) and (2) imply

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

- **case** $e_1\ e_2$ for $\mathcal{G}^R$: By induction, ($\mathcal{G}$.18) implies $S_1\Gamma \vdash e_2 : S_1\beta$. By Lemma A1.1, we can apply S_3S_2 to both sides:

  $$S_3S_2S_1\Gamma \vdash e_2 : S_3S_2S_1\beta. \quad (3)$$

  By induction, ($\mathcal{G}$.19) implies $S_2S_1\Gamma \vdash e_1 : S_2\theta$. By Lemma A1.1, we can apply S_3 to both sides:

  $$S_3S_2S_1\Gamma \vdash e_1 : S_3S_2\theta.$$

  Because $S_3S_2\theta = S_3S_2S_1(\beta \to \rho)$ by ($\mathcal{G}$.20),

  $$S_3S_2S_1\Gamma \vdash e_1 : S_3S_2S_1(\beta \to \rho). \quad (4)$$

  Hence by (APP), (3) and (4) imply

  $$S_3S_2S_1\Gamma \vdash e_1\ e_2 : S_3S_2S_1\rho.$$

- **case** `let` x=e_1 `in` e_2 : Let $S_2' = S_2\{\vec{\beta}/\vec{\alpha}\}$, where $\vec{\alpha} = \mathit{ftv}(S_1\beta) \setminus \mathit{ftv}(S_1\Gamma)$, $\vec{\beta}$ are new type variables, and β 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 A1.1, we can apply S_2' to both sides:

  $$S_2'S_1\Gamma \vdash e_1 : S_2'S_1\beta. \quad (5)$$

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

  $$S_2S_1\Gamma + x{:}\ S_2\mathit{Clos}_{S_1\Gamma}(S_1\beta) \vdash e_2 : S_2\theta. \quad (6)$$

Note that $S_2S_1\Gamma = S_2'S_1\Gamma$ because S_2' differs from S_2 only on non-free variables of $S_1\Gamma$, and that $S_2 Clos_{S_1\Gamma}(S_1\beta) = Clos_{S_2'S_1\Gamma}(S_2'S_1\beta)$ by Lemma A1.3. Thus (6) is

$$S_2'S_1\Gamma + x\colon Clos_{S_2'S_1\Gamma}(S_2'S_1\beta) \vdash e_2 : S_2\theta. \quad (7)$$

Hence by (LET), (5) and (7) imply $S_2'S_1\Gamma \vdash \texttt{let}\ x{=}e_1\ \texttt{in}\ e_2 : S_2\theta$; that is,

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

By Lemma A1.1, we can apply S_3 to both sides:

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

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

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

- **case** $\texttt{fix}\ f\ \lambda x.e$: By induction, ($\mathcal{G}$.16) implies $S_2S_1\Gamma_1 + x\colon S_2S_1\beta_1 \vdash e : S_2S_1\beta_2$. By (FN),

$$S_2S_1\Gamma_1 \vdash \lambda x.e : S_2S_1(\beta_1 \to \beta_2).$$

By Lemma A1.1, we can apply S_3 to both sides:

$$S_3S_2S_1\Gamma_1 \vdash \lambda x.e : S_3S_2S_1(\beta_1 \to \beta_2).$$

Because $S_1(\beta_1 \to \beta_2) = S_1\theta_2$ by ($\mathcal{G}$.15),

$$S_3S_2S_1\Gamma_1 \vdash \lambda x.e : S_3S_2S_1\theta_2.$$

Because $\Gamma_1 = \Gamma + f\colon \theta_1$ by ($\mathcal{G}$.14),

$$S_3S_2S_1\Gamma + f\colon S_3S_2S_1\theta_1 \vdash \lambda x.e : S_3S_2S_1\theta_2.$$

Because $S_3S_2S_1\theta_1 = S_3S_2S_1\theta_2 = S_3S_2S_1\rho$ by ($\mathcal{G}$.17),

$$S_3S_2S_1\Gamma + f\colon S_3S_2S_1\rho \vdash \lambda x.e : S_3S_2S_1\rho.$$

Hence by (FIX),

$$S_3S_2S_1\Gamma \vdash \texttt{fix}\ f\ \lambda x.e : S_3S_2S_1\rho.$$

■

§A2 Completeness Proof

Theorem 2.2 (Completeness)
Let e be an expression, and let Γ be a type environment. If there exist a type ρ and a substitution P such that $P\Gamma \vdash e : P\rho$, then $\mathcal{G}(\Gamma, e, \rho)$ succeeds with S and there exists a substitution R such that $P\!\restriction_{New} = (RS)\!\restriction_{New}$ where *New* is the set of new type variables used by $\mathcal{G}(\Gamma, e, \rho)$. The theorem also holds for $\mathcal{G}^R$.

The completeness proof uses Lemmas A2.1–A2.5.

Lemma A2.1 (Lee and Yi[10])
Let S be a substitution, Γ be a type environment, and τ be a type. Then $S Clos_\Gamma(\tau) \succ Clos_{S\Gamma}(S\tau)$.

Lemma A2.2 (Damas and Milner[6])
Let Γ and Γ' be type environments such that $\Gamma \succ \Gamma'$. If $\Gamma' \vdash e : \tau$, then $\Gamma \vdash e : \tau$.

Lemma A2.3 (Milner[14])
Let R and S be substitutions and τ be a type. Then

- $itv(RS) \subseteq itv(R) \cup itv(S)$ and
- $ftv(S\tau) \subseteq ftv(\tau) \cup itv(S)$.

Lemma A2.4
If $S = \mathcal{G}(\Gamma, e, \rho)$ then $itv(S) \subseteq ftv(\Gamma) \cup ftv(\rho) \cup New$, where New is the set of new type variables used by $\mathcal{G}(\Gamma, e, \rho)$. The lemma also hols for $\mathcal{G}^R$.

Proof
See Appendix Section A3. ■

Lemma A2.5 (Lee and Yi[10])
If $itv(S) \cap V = \emptyset$, then $(RS)|_V = R|_V S$.

Proof of Theorem 2.2
We prove by structural induction on e, and we prove for $\mathcal{G}$ and $\mathcal{G}^R$ simultaneously. For a rigorous treatment of new type variables, we assume that every new type variable used throughout algorithm $\mathcal{G}$ is distinct from each other, and that the set New of new type variables used by each call $\mathcal{G}(\Gamma, e, \rho)$ satisfies $New \cap (ftv(\Gamma) \cup ftv(\rho)) = \emptyset$. Moreover, let us rephrase the part of the algorithm definition that whenever we use $\theta \geq \rho$ in $\mathcal{G}$, the substitution G for $G\theta = \rho$ is such that $supp(G) = ftv(\theta) \setminus ftv(\rho)$ and has only new type variables.

- **case** $()$ **and** x : The same as the proof for $\mathcal{M}$ in Lee and Yi's.[10]
- **case** $\lambda x.e$: Let the given judgment be $P\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2$ where $\tau_1 \to \tau_2 = P\rho$, and $New = \{\beta_1, \beta_2\} \cup supp(G) \cup New_1$ where β_1 and β_2 are new type variables used at ($\mathcal{G}$.3), G is the substitution for $\theta \geq \rho$ at ($\mathcal{G}$.3), and New_1 is the set of new type variables used by $\mathcal{G}(S_1\Gamma + x\colon S_1\beta_1, e, S_1\beta_2)$ at ($\mathcal{G}$.4).
First, we prove the unification $\mathcal{U}(\beta_1 \to \beta_2, \theta)$ at ($\mathcal{G}$.3) succeeds. Let $P' = (PG)|_{\{\beta_1,\beta_2\}} \cup \{\tau_1/\beta_1, \tau_2/\beta_2\}$. Then P' unifies $\beta_1 \to \beta_2$ and θ because

$$\begin{array}{rcll} P'\theta & = & PG\theta & \text{because the new } \beta_1, \beta_2 \notin ftv(\theta) \\ & = & P\rho & \text{by the definition of } G \\ & = & \tau_1 \to \tau_2 & \text{by the assumption} \\ & = & P'(\beta_1 \to \beta_2) & \text{by the definition of } P'. \end{array}$$

Thus by Theorem 1.1, the unification at ($\mathcal{G}$.3) succeeds with S_1 such that for a substitution R_1,

$$R_1 S_1 = P'. \tag{8}$$

By the (FN) rule, the given judgment implies

$$P\Gamma + x\colon \tau_1 \vdash e : \tau_2. \tag{9}$$

To apply induction to $\mathcal{G}(S_1\Gamma + x\colon S_1\beta_1, e, S_1\beta_2)$ at ($\mathcal{G}$.4) and (9), we must prove that there exists a substitution P_1 such that $\tau_2 = P_1(S_1\beta_2)$ and $P\Gamma + x\colon \tau_1 = P_1(S_1\Gamma + x\colon S_1\beta_1)$. Such P_1 is R_1 at (8) because

$$\begin{array}{rcll} R_1(S_1\beta_2) & = & P'\beta_2 & \text{by (8)} \\ & = & \tau_2 & \text{by the definition of } P' \end{array}$$

and

$$\begin{aligned} & R_1(S_1\Gamma + x\colon S_1\beta_1) \\ & = P'(\Gamma + x\colon \beta_1) && \text{by (8)} \\ & = PG\Gamma + x\colon \tau_1 && \text{because the new } \beta_1, \beta_2 \notin \mathit{ftv}(\Gamma) \\ & = P\Gamma + x\colon \tau_1 && \text{because } \mathit{supp}(G) \cap \mathit{ftv}(\Gamma) = \emptyset. \end{aligned}$$

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

$$(R_2S_2)\restriction_{New_1} = R_1\restriction_{New_1}. \tag{10}$$

Note that

$$\begin{aligned} \mathit{itv}(S_1) & \subseteq \{\beta_1, \beta_2\} \cup \mathit{ftv}(\theta) && \text{by Theorem 1.1} \\ & \subseteq \{\beta_1, \beta_2\} \cup \mathit{ftv}(\rho) \cup \mathit{supp}(G) \end{aligned}$$

because $\mathit{supp}(G) = \mathit{ftv}(\theta) \setminus \mathit{ftv}(\rho)$, and thus by the definition of $\mathcal{G}$,

$$New_1 \cap \mathit{itv}(S_1) = \emptyset. \tag{11}$$

Then

$$\begin{aligned} (R_2S_2S_1)\restriction_{New_1} & = (R_2S_2)\restriction_{New_1} S_1 && \text{by Lemma A2.5 and (11)} \\ & = R_1\restriction_{New_1} S_1 && \text{by (10)} \\ & = (R_1S_1)\restriction_{New_1} && \text{by Lemma A2.5 and (11)} \\ & = P'\restriction_{New_1} && \text{by (8).} \end{aligned} \tag{12}$$

Now we prove the unification $\mathcal{U}(S_2S_1\theta, S_2S_1\rho)$ at ($\mathcal{G}$.5) succeeds. R_2 unifies $S_2S_1\theta$ and $S_2S_1\rho$ because

$$\begin{aligned} & R_2(S_2S_1\theta) \\ & = P'\theta && \text{by (12) and because } \mathit{ftv}(\theta) \cap New_1 = \emptyset \\ & = PG\theta && \text{because the new } \beta_1, \beta_2 \notin \mathit{ftv}(\theta) \\ & = P\rho && \text{by the definition of } G \\ & = PG\rho && \text{because } \mathit{ftv}(\rho) \cap \mathit{supp}(G) = \emptyset \\ & = P'\rho && \text{because the new } \beta_1, \beta_2 \notin \mathit{ftv}(\rho) \\ & = R_2(S_2S_1\rho) && \text{by (12) and because } \mathit{ftv}(\rho) \cap New_1 = \emptyset. \end{aligned}$$

Thus the unification at ($\mathcal{G}$.5) succeeds with S_3 such that for a substitution R_3,

$$R_3S_3 = R_2. \tag{13}$$

Hence $\mathcal{G}(\Gamma, \lambda x.e, \rho)$ succeeds with $S_3S_2S_1$, and $(R_3S_3S_2S_1)\restriction_{New} = P\restriction_{New}$ because

$$\begin{aligned} & (R_3S_3S_2S_1)\restriction_{New} \\ & = (R_2S_2S_1)\restriction_{New} && \text{by (13)} \\ & = P'\restriction_{New} && \text{by (12)} \\ & = P\restriction_{New} && \text{because } \mathit{supp}(G) \cup \{\beta_1, \beta_2\} \subseteq New. \end{aligned}$$

- **case** $e_1\ e_2$ **for** $\mathcal{G}$: Let the given judgment be $P\Gamma \vdash e_1\ e_2 : P\rho$, and $New = \{\beta\} \cup \mathit{supp}(G_1) \cup \mathit{supp}(G_2) \cup \mathit{supp}(G_3) \cup New_1 \cup New_2$, where β is the new type variable used at ($\mathcal{G}$.6), G_1, G_2 and G_3 are respectively the substitutions for $\theta_1 \geq \beta \rightarrow \rho$ at ($\mathcal{G}$.6), $\theta_2 \geq S_1(\beta \rightarrow \rho)$ at ($\mathcal{G}$.7), and $\theta_3 \geq S_2S_1\beta$ at ($\mathcal{G}$.8), and New_1 and New_2 are respectively the sets of the new type variables used by $\mathcal{G}(\Gamma, e_1, \theta_1)$ at ($\mathcal{G}$.6) and $\mathcal{G}(S_2S_1\Gamma, e_2, \theta_3)$ at ($\mathcal{G}$.8).

By the (APP) rule, there exists a type τ such that

$$P\Gamma \vdash e_1 : \tau \to P\rho \tag{14}$$

and

$$P\Gamma \vdash e_2 : \tau. \tag{15}$$

First, we prove $\mathcal{G}(\Gamma, e_1, \theta_1)$ at ($\mathcal{G}$.6) succeeds by induction. Let $P' = P|_{\{\beta\}} \cup \{\tau/\beta\}$. Then

$$\begin{array}{rcll} P'G_1\theta_1 & = & P'(\beta \to \rho) & \text{by the definition of } G_1 \\ & = & \tau \to P\rho & \text{because the new } \beta \notin \mathit{ftv}(\rho) \end{array}$$

and $P'G_1\Gamma = P\Gamma$ because $\mathit{ftv}(\Gamma) \cap (\mathit{supp}(G_1) \cup \{\beta\}) = \emptyset$. Hence, applying induction to $\mathcal{G}(\Gamma, e_1, \theta_1)$ at ($\mathcal{G}$.6) and (14), there exists a substitution R_1 such that

$$(R_1S_1)|_{New_1} = (P'G_1)|_{New_1}. \tag{16}$$

Then R_1G_2 unifies $S_1\theta_1$ and θ_2 at ($\mathcal{G}$.7) because, by noting that

$$\begin{array}{ll} \mathit{ftv}(S_1\theta_1) \cap \mathit{supp}(G_2) & \\ \subseteq (\mathit{itv}(S_1) \cup \mathit{ftv}(\theta_1)) \cap \mathit{supp}(G_2) & \text{by Lemma A2.3} \\ \subseteq (\mathit{ftv}(\Gamma) \cup New_1 \cup \mathit{ftv}(\theta_1)) \cap \mathit{supp}(G_2) & \text{by Lemma A2.4} \\ = \emptyset, & \end{array} \tag{17}$$

$$\begin{array}{ll} R_1G_2(S_1\theta_1) & \\ = R_1S_1\theta_1 & \text{by (17)} \\ = P'G_1\theta_1 & \text{by (16) and because } \mathit{ftv}(\theta_1) \cap New_1 = \emptyset \\ = P'(\beta \to \rho) & \text{by the definition of } G_1 \\ = P'G_1(\beta \to \rho) & \text{because } \mathit{ftv}(\beta \to \rho) \cap \mathit{supp}(G_1) = \emptyset \\ = R_1S_1(\beta \to \rho) & \text{by (16) and because } \mathit{ftv}(\beta \to \rho) \cap New_1 = \emptyset \\ = R_1G_2(\theta_2) & \text{by the definition of } G_2. \end{array}$$

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

$$\begin{array}{ll} (R_2S_2S_1)|_{\mathit{supp}(G_2) \cup New_1} & \\ = (R_1G_2S_1)|_{\mathit{supp}(G_2) \cup New_1} & \\ = (R_1S_1)|_{\mathit{supp}(G_2) \cup New_1} & \text{because } \mathit{supp}(G_2) \cap \mathit{itv}(S_1) = \emptyset \\ & \quad \text{by Lemma A2.4} \\ = (P'G_1)|_{\mathit{supp}(G_2) \cup New_1} & \text{by (16).} \end{array} \tag{18}$$

In order to apply induction to $\mathcal{G}(S_2S_1\Gamma, e_2, \theta_3)$ at ($\mathcal{G}$.8) and (15), 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 $\mathcal{G}$,

$$\mathit{supp}(G_3) \cap \mathit{ftv}(S_2S_1\Gamma) = \emptyset \tag{19}$$

because

$$\begin{array}{ll} \mathit{ftv}(S_2S_1\Gamma) & \\ \subseteq \mathit{itv}(S_2) \cup \mathit{itv}(S_1) \cup \mathit{ftv}(\Gamma) & \text{by Lemma A2.3} \\ \subseteq \mathit{ftv}(\theta_2) \cup \mathit{ftv}(\theta_1) \cup New_1 \cup \mathit{ftv}(\Gamma) & \text{by Theorem 1.1 and Lemma A2.4.} \end{array}$$

Thus

$$\begin{array}{ll} R_2G_3(S_2S_1\Gamma) & \\ = R_2S_2S_1\Gamma & \text{by (19)} \\ = P'G_1\Gamma & \text{by (18) and} \\ & \text{because } \mathit{ftv}(\Gamma) \cap (\mathit{supp}(G_2) \cup \mathit{New}_1) = \emptyset \\ = P\Gamma & \text{because } \mathit{ftv}(\Gamma) \cap (\{\beta\} \cup \mathit{supp}(G_1)) = \emptyset. \end{array}$$

Second,

$$\begin{array}{ll} R_2G_3(\theta_3) & \\ = R_2S_2S_1\beta & \text{by the definition of } G_3 \\ = P'G_1\beta & \text{by (18) and because } \beta \notin \mathit{supp}(G_2) \cup \mathit{New}_1 \\ = P'\beta & \text{because } \beta \notin \mathit{supp}(G_1) \\ = \tau & \text{by the definition of } P'. \end{array}$$

Thus by induction, ($\mathcal{G}$.8) succeeds with S_3 such that for a substitution R_3,

$$(R_3S_3)|_{New_2} = (R_2G_3)|_{New_2}. \tag{20}$$

Moreover, note that

$$(R_3S_3)|_{New_2 \cup supp(G_3)} = R_2|_{New_2 \cup supp(G_3)}. \tag{21}$$

Then R_3 unifies $S_3S_2S_1\theta_1$ and $S_3S_2S_1(\beta \to \rho)$ at ($\mathcal{G}$.9) because

$$\begin{array}{ll} R_3S_3S_2S_1\theta_1 & \\ = R_2S_2S_1\theta_1 & \text{by (21) and} \\ & \text{because } \mathit{ftv}(\theta_1) \cap (\mathit{New}_2 \cup \mathit{supp}(G_3)) = \emptyset \\ = P'G_1\theta_1 & \text{by (18) and} \\ & \text{because } \mathit{ftv}(\theta_1) \cap (\mathit{New}_1 \cup \mathit{supp}(G_2)) = \emptyset \\ = P'(\beta \to \rho) & \text{by the definition of } G_1 \\ = P'G_1(\beta \to \rho) & \text{because } \mathit{ftv}(\beta \to \rho) \cap \mathit{supp}(G_1) = \emptyset \\ = R_2S_2S_1(\beta \to \rho) & \text{by (18) and} \\ & \text{because } \mathit{ftv}(\beta \to \rho) \cap (\mathit{New}_1 \cup \mathit{supp}(G_2)) = \emptyset \\ = R_3S_3S_2S_1(\beta \to \rho) & \text{by (21) and} \\ & \text{because } \mathit{ftv}(\beta \to \rho) \cap (\mathit{New}_2 \cup \mathit{supp}(G_3)) = \emptyset. \end{array}$$

Thus the unification at ($\mathcal{G}$.9) succeeds with S_4 such that for a substitution R_4,

$$R_4S_4 = R_3. \tag{22}$$

Finally, R_4 unifies $S_4S_3\theta_3$ and $S_4S_3S_2S_1\beta$ at ($\mathcal{G}$.10) because

$$\begin{array}{ll} R_4(S_4S_3\theta_3) & \\ = R_3S_3\theta_3 & \text{by (22)} \\ = R_2G_3\theta_3 & \text{by (20) and because } \mathit{ftv}(\theta_3) \cap \mathit{New}_2 = \emptyset \\ = R_2S_2S_1\beta & \text{by the definition of } G_3 \\ = R_4(S_4S_3S_2S_1\beta) & \text{by (21) and (22), and} \\ & \text{because } \beta \notin \mathit{New}_2 \cup \mathit{supp}(G_3). \end{array}$$

Thus the unification at ($\mathcal{G}$.10) succeeds with S_5 such that for a substitution R_5,

$$R_5S_5 = R_4. \tag{23}$$

Hence $\mathcal{G}(\Gamma, e_1\ e_2, \rho)$ succeeds with $S_5S_4S_3S_2S_1$.

Now we prove the rest that $(R_5S_5S_4S_3S_2S_1)|_{New} = P|_{New}$. Note that, by Lemma A2.3 and A2.4 and Theorem 1.1, $itv(S_2S_1) \subseteq ftv(\Gamma) \cup ftv(\theta_1) \cup ftv(\theta_2) \cup New_1$, hence by the definition of $\mathcal{G}$,

$$itv(S_2S_1) \cap (New_2 \cup supp(G_3)) = \emptyset. \tag{24}$$

Therefore

$$\begin{array}{ll} (R_5S_5S_4S_3S_2S_1)|_{New} & \\ = (R_4S_4S_3S_2S_1)|_{New} & \text{by (23)} \\ = (R_3S_3S_2S_1)|_{New} & \text{by (22)} \\ = ((R_3S_3)|_{New_2 \cup supp(G_3)}S_2S_1)|_{New} & \text{by Lemma A2.5 and (24)} \\ = (R_2|_{New_2 \cup supp(G_3)}S_2S_1)|_{New} & \text{by (21)} \\ = (R_2S_2S_1)|_{New} & \text{by Lemma A2.5 and (24)} \\ = (P'G_1)|_{New} & \text{by (18)} \\ = P|_{New} & \text{because } (\{\beta\} \cup supp(G_1)) \subseteq New. \end{array}$$

- **case** $e_1\ e_2$ for $\mathcal{G}^R$:

Let the given judgment be $P\Gamma \vdash e_1\ e_2 : P\rho$, and $New = \{\beta\} \cup supp(G) \cup New_1 \cup New_2$, where β is the new type variable used at ($\mathcal{G}$.18), G is the substitution for $\theta \geq S_1(\beta \to \rho)$ at ($\mathcal{G}$.19), and New_1 and New_2 are respectively the sets of the new type variables used by $\mathcal{G}(\Gamma, e_2, \beta)$ at ($\mathcal{G}$.18) and $\mathcal{G}(S_1\Gamma, e_1, \theta)$ at ($\mathcal{G}$.19).

By the (APP) rule, there exists a type τ such that

$$P\Gamma \vdash e_1 : \tau \to P\rho \tag{25}$$

and

$$P\Gamma \vdash e_2 : \tau. \tag{26}$$

First, we prove that $\mathcal{G}(\Gamma, e_2, \beta)$ at ($\mathcal{G}$.18) succeeds by induction. Let $P' = P|_{\{\beta\}} \cup \{\tau/\beta\}$. Then $P'\beta = \tau$ and $P'\Gamma = P\Gamma$ because $\beta \notin ftv(\Gamma)$. Hence by induction, $\mathcal{G}(\Gamma, e_2, \beta)$ at ($\mathcal{G}$.18) and (26) imply that there exists a substitution R_1 such that

$$(R_1S_1)|_{New_1} = P'|_{New_1}. \tag{27}$$

Moreover, note that

$$(R_1S_1)|_{\{\beta\} \cup New_1} = P|_{\{\beta\} \cup New_1}. \tag{28}$$

In order to apply induction to $\mathcal{G}(S_1\Gamma, e_1, \theta)$ at ($\mathcal{G}$.19) and (25), we must find a substitution P_1 such that $P_1S_1\Gamma = P\Gamma$ and $P_1\theta = \tau \to P\rho$. Such P_1 is R_1G because

$$\begin{array}{ll} R_1G(S_1\Gamma) & \\ = R_1S_1\Gamma & \text{because } supp(G) \cap ftv(S_1\Gamma) = \emptyset \\ & \text{by Lemma A2.3 and A2.4} \\ = P\Gamma & \text{by (28) and because } ftv(\Gamma) \cap (\{\beta\} \cup New_1) = \emptyset. \end{array}$$

and

$$\begin{array}{ll} R_1G(\theta) & \\ = R_1S_1(\beta \to \rho) & \text{by the definition of } G \\ = P'(\beta \to \rho) & \text{by (27) and because } ftv(\beta \to \rho) \cap New_1 = \emptyset \\ = \tau \to P'\rho & \text{by the definition of } P' \\ = \tau \to P\rho & \text{because } \beta \notin ftv(\rho). \end{array}$$

Then by induction, ($\mathcal{G}$.19) succeeds with S_2 such that for a substitution R_2,

$$(R_2S_2)|_{New_2} = (R_1G)|_{New_2}. \tag{29}$$

Moreover, note that

$$(R_2S_2)|_{supp(G)\cup New_2} = R_1|_{supp(G)\cup New_2}. \tag{30}$$

Then R_2 unifies $S_2\theta$ and $S_2S_1(\beta \to \rho)$ at ($\mathcal{G}$.20) because

$$\begin{array}{ll} R_2S_2\theta & \\ = R_1G\theta & \text{by (29) and } ftv(\theta) \cap New_2 = \emptyset \\ = R_1S_1(\beta \to \rho) & \text{by the definition of } G \\ = R_2S_2S_1(\beta \to \rho) & \text{by (30) and because, by Lemma A2.3 and A2.4,} \\ & ftv(S_2S_1(\beta \to \rho)) \cap (supp(G) \cap New_2) = \emptyset. \end{array}$$

Thus the unification at ($\mathcal{G}$.20) succeeds with S_3 such that for a substitution R_3,

$$R_3S_3 = R_2. \tag{31}$$

Hence $\mathcal{G}(\Gamma, e_1\ e_2, \rho)$ succeeds with $S_3S_2S_1$.

Now we prove the rest that $(R_3S_3S_2S_1)|_{New} = P|_{New}$. Note that, by Lemma A2.4, $itv(S_1) \subseteq ftv(\Gamma) \cup \{\beta\} \cup New_1$, hence by the definition of $\mathcal{G}$,

$$itv(S_1) \cap (supp(G) \cup New_2) = \emptyset. \tag{32}$$

Therefore

$$\begin{array}{ll} (R_3S_3S_2S_1)|_{New} & \\ = (R_2S_2S_1)|_{New} & \text{by (31)} \\ = ((R_2S_2)|_{supp(G)\cup New_2}S_1)|_{New} & \text{by Lemma A2.5 and (32)} \\ = (R_1|_{supp(G)\cup New_2}S_1)|_{New} & \text{by (30)} \\ = (R_1S_1)|_{New} & \text{by Lemma A2.5 and (32)} \\ = P|_{New} & \text{by (28).} \end{array}$$

- **case** `let` x=e_1 `in` e_2 : Let the given judgment be $P\Gamma \vdash$ `let` x=e_1 `in` $e_2 : P\rho$, and $New = \{\beta\} \cup supp(G) \cup New_1 \cup New_2$, where β is the new type variable introduced at ($\mathcal{G}$.11), G is the substitution for $\theta \geq S_1\rho$ at ($\mathcal{G}$.12), and New_1 and New_2 are respectively the sets of new type variables used by $\mathcal{G}(\Gamma, e_1, \beta)$ at ($\mathcal{G}$.11) and $\mathcal{G}(S_1\Gamma + x{:}\ Clos_{S_1\Gamma}(S_1\beta), e_2, \theta)$ at ($\mathcal{G}$.12).
  By the (LET) rule, there exists a type τ such that

$$P\Gamma \vdash e_1 : \tau \tag{33}$$

and

$$P\Gamma + x{:}\ Clos_{P\Gamma}(\tau) \vdash e_2 : P\rho. \tag{34}$$

Let $P' = P\nmid_{\{\beta\}} \cup \{\tau/\beta\}$. Then $P'\beta = \tau$ and $P'\Gamma = P\Gamma$ because $\beta \notin ftv(\Gamma)$. Hence by induction, $\mathcal{G}(\Gamma, e_1, \beta)$ at ($\mathcal{G}$.11) and (33) imply that there exists a substitution R_1 such that

$$(R_1 S_1)\nmid_{New_1} = P'\nmid_{New_1}. \tag{35}$$

Moreover,

$$(R_1 S_1)\nmid_{\{\beta\}\cup New_1} = P\nmid_{\{\beta\}\cup New_1}. \tag{36}$$

Note that

$$\begin{array}{lll} & R_1 G(S_1\Gamma) & \\ = & R_1 S_1 \Gamma & \text{because } supp(G) \cap ftv(S_1\Gamma) = \emptyset \\ & & \text{by Lemma A2.3 and A2.4} \\ = & P\Gamma & \text{by (36) and because } ftv(\Gamma) \cap (\{\beta\} \cup New_1) = \emptyset, \end{array}$$

and

$$\begin{array}{lll} & R_1 G(Clos_{S_1\Gamma}(S_1\beta)) & \\ \succ & Clos_{R_1 G S_1 \Gamma}(R_1 G S_1 \beta) & \text{by Lemma A2.1} \\ = & Clos_{P\Gamma}(R_1 S_1 \beta) & \text{because } supp(G) \cap ftv(S_1\beta) = \emptyset \\ & & \text{by Lemma A2.3 and A2.4} \\ = & Clos_{P\Gamma}(P'\beta) & \text{by (35) and because } \beta \notin New_1 \\ = & Clos_{P\Gamma}(\tau) & \text{by the definition of } P'; \end{array}$$

that is, $R_1 G(S_1\Gamma + x\colon Clos_{S_1\Gamma}(S_1\beta)) \succ P\Gamma + x\colon Clos_{P\Gamma}(\tau)$. Then by Lemma A2.2 and (34),

$$R_1 G(S_1\Gamma + x\colon Clos_{S_1\Gamma}(S_1\beta)) \vdash e_2 : P\rho. \tag{37}$$

In order to apply induction to $\mathcal{G}(S_1\Gamma + x\colon Clos_{S_1\Gamma}(S_1\beta), e_2, \theta)$ at ($\mathcal{G}$.12) and (37), we have to prove that $R_1 G\theta = P\rho$:

$$\begin{array}{lll} & R_1 G(\theta) & \\ = & R_1 S_1 \rho & \text{by the definition of } G \\ = & P\rho & \text{by (36) and because } ftv(\rho) \cap (\{\beta\} \cup New_1) = \emptyset. \end{array}$$

Thus by induction, $\mathcal{G}(S_1\Gamma + x\colon Clos_{S_1\Gamma}(S_1\beta), e_2, \theta)$ at ($\mathcal{G}$.12) succeeds with S_2 such that for a substitution R_2,

$$(R_2 S_2)\nmid_{New_2} = (R_1 G)\nmid_{New_2}. \tag{38}$$

Moreover, note that

$$(R_2 S_2)\nmid_{supp(G)\cup New_2} = R_1\nmid_{supp(G)\cup New_2}. \tag{39}$$

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

$$\begin{array}{lll} & R_2(S_2\theta) & \\ = & R_1 G\theta & \text{by (38) and because } ftv(\theta) \cap New_2 = \emptyset \\ = & R_1 S_1 \rho & \text{by the definition of } G \\ = & R_2(S_2 S_1 \rho) & \text{by (39) and because, by Lemma A2.3 and A2.4,} \\ & & ftv(S_1\rho) \cap (supp(G) \cup New_2) = \emptyset. \end{array}$$

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

$$R_3 S_3 = R_2. \tag{40}$$

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

Now we prove the rest that $(R_3S_3S_2S_1)\restriction_{New} = P\restriction_{New}$. Note that, by Lemma A2.4, $itv(S_1) \subseteq ftv(\Gamma) \cup \{\beta\} \cup New_1$, hence by the definition of $\mathcal{G}$,

$$itv(S_1) \cap (supp(G) \cup New_2) = \emptyset. \tag{41}$$

Therefore

$$\begin{array}{ll} (R_3S_3S_2S_1)\restriction_{New} & \\ = (R_2S_2S_1)\restriction_{New} & \text{by (40)} \\ = ((R_2S_2)\restriction_{supp(G)\cup New_2}S_1)\restriction_{New} & \text{by Lemma A2.5 and (41)} \\ = (R_1\restriction_{supp(G)\cup New_2}S_1)\restriction_{New} & \text{by (39)} \\ = (R_1S_1)\restriction_{New} & \text{by Lemma A2.5 and (41)} \\ = P\restriction_{New} & \text{by (36).} \end{array}$$

- **case** `fix` $f\ \lambda x.e$: Let the given judgment be $P\Gamma \vdash$ `fix` $f\ \lambda x.e : P\rho$ where $P\rho = \tau_1 \to \tau_2$ and $New = \{\beta_1, \beta_2\} \cup supp(G_1) \cup supp(G_2) \cup New'$ where β_1 and β_2 are new type variables used at ($\mathcal{G}$.15), G_1 and G_2 are substitutions for $\theta_1 \geq \rho$ at ($\mathcal{G}$.14) and $\theta_2 \geq \theta_1$ at ($\mathcal{G}$.15), and New' is the set of new type variables used by $\mathcal{G}(S_1\Gamma_1 + x\colon S_1\beta_1, e, S_1\beta_2)$ at ($\mathcal{G}$.15).
  By the (FIX) rule, $P\Gamma + f\colon P\rho \vdash \lambda x.e : P\rho$. Because $(supp(G_1) \cup supp(G_2)) \cap ftv(\Gamma) = \emptyset$ and $\rho = G_1\theta_1 = G_1G_2\theta_2$,

  $$PG_1G_2\Gamma + f\colon PG_1\theta_1 \vdash \lambda x.e : PG_1G_2\theta_2.$$

  Because $\Gamma_1 = \Gamma + f : \theta_1$ by ($\mathcal{G}$.14), and $ftv(\theta_1) \cap supp(G_2) = \emptyset$,

  $$PG_1G_2\Gamma_1 \vdash \lambda x.e : PG_1G_2\theta_2. \tag{42}$$

  First, we prove the unification $\mathcal{U}(\beta_1 \to \beta_2, \theta_2)$ at ($\mathcal{G}$.15) succeeds. Let $P' = (PG_1G_2)\restriction_{\{\beta_1,\beta_2\}} \cup \{\tau_1/\beta_1, \tau_2/\beta_2\}$. Then P' unifies $\beta_1 \to \beta_2$ and θ_2 because

  $$\begin{array}{rll} P'\theta_2 = & PG_1G_2\theta_2 & \text{because the new } \beta_1, \beta_2 \notin ftv(\theta_2) \\ = & PG_1\theta_1 & \text{by the definition of } G_2 \\ = & P\rho & \text{by the definition of } G_1 \\ = & \tau_1 \to \tau_2 & \text{by the assumption} \\ = & P'(\beta_1 \to \beta_2) & \text{by the definition of } P'. \end{array} \tag{43}$$

  Thus by Theorem 1.1, the unification at ($\mathcal{G}$.15) succeeds with S_1 such that for a substitution R_1,

  $$R_1S_1 = P'. \tag{44}$$

  By the (FN) rule and because $PG_1G_2\theta_2 = \tau_1 \to \tau_2$ by (43), (42) implies

  $$PG_1G_2\Gamma_1 + x\colon \tau_1 \vdash e : \tau_2. \tag{45}$$

  To apply induction to $\mathcal{G}(S_1\Gamma_1 + x\colon S_1\beta_1, e, S_1\beta_2)$ at ($\mathcal{G}$.16) and (45), we must prove that there exists a substitution P_1 such that $\tau_2 = P_1(S_1\beta_2)$ and $PG_1G_2\Gamma_1 + x\colon \tau_1 = P_1(S_1\Gamma_1 + x\colon S_1\beta_1)$. Such P_1 is R_1 at (44) because

  $$\begin{array}{rcll} R_1(S_1\beta_2) & = & P'\beta_2 & \text{by (44)} \\ & = & \tau_2 & \text{by the definition of } P' \end{array}$$

  and

$$\begin{aligned}&R_1(S_1\Gamma_1 + x\colon S_1\beta_1)\\ &= P'(\Gamma_1 + x\colon \beta_1) \quad \text{by (44)}\\ &= PG_1G_2\Gamma_1 + x\colon \tau_1 \quad \text{because the new } \beta_1, \beta_2 \notin \mathit{ftv}(\Gamma_1).\end{aligned}$$

Thus by induction, $\mathcal{G}(S_1\Gamma_1 + x\colon S_1\beta_1, e, S_1\beta_2)$ at ($\mathcal{G}$.16) succeeds with S_2 such that for a substitution R_2,

$$(R_2S_2)\restriction_{New'} = R_1\restriction_{New'}. \tag{46}$$

Note that, because $\mathit{supp}(G_1) = \mathit{ftv}(\theta_1)\backslash\mathit{ftv}(\rho)$ and $\mathit{supp}(G_2) = \mathit{ftv}(\theta_2)\backslash\mathit{ftv}(\theta_1)$,

$$\begin{aligned}&\mathit{itv}(S_1)\\ &\subseteq \{\beta_1, \beta_2\} \cup \mathit{ftv}(\theta_2) \quad \text{by Theorem 1.1}\\ &\subseteq \{\beta_1, \beta_2\} \cup \mathit{ftv}(\theta_1) \cup \mathit{supp}(G_2)\\ &\subseteq \{\beta_1, \beta_2\} \cup \mathit{ftv}(\rho) \cup \mathit{supp}(G_1) \cup \mathit{supp}(G_2)\end{aligned}$$

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

$$New' \cap \mathit{itv}(S_1) = \emptyset. \tag{47}$$

Then

$$\begin{aligned}(R_2S_2S_1)\restriction_{New'} &= (R_2S_2)\restriction_{New'}S_1 && \text{by Lemma A2.5 and (47)}\\ &= R_1\restriction_{New'}S_1 && \text{by (46)}\\ &= (R_1S_1)\restriction_{New'} && \text{by Lemma A2.5 and (47)}\\ &= P'\restriction_{New'} && \text{by (44).}\end{aligned} \tag{48}$$

Now we prove the unification $\mathcal{U}(S_2S_1\theta_1, S_2S_1\theta_2, S_2S_1\rho)$ at ($\mathcal{G}$.16) succeeds. R_2 unifies $S_2S_1\theta_1$, $S_2S_1\theta_2$, and $S_2S_1\rho$ because

$$\begin{aligned}&R_2(S_2S_1\theta_2)\\ &= P'\theta_2 && \text{by (48) and because } \mathit{ftv}(\theta_2) \cap New' = \emptyset\\ &= PG_1G_2\theta_2 && \text{because the new } \beta_1, \beta_2 \notin \mathit{ftv}(\theta_2)\\ &= PG_1\theta_1 && \text{by the definition of } G_2\\ &= PG_1G_2\theta_1 && \text{because } \mathit{ftv}(\theta_1) \cap \mathit{supp}(G_2) = \emptyset\\ &= P'\theta_1 && \text{because the new } \beta_1, \beta_2 \notin \mathit{ftv}(\theta_1)\\ &= R_2(S_2S_1\theta_1) && \text{by (48) and because } \mathit{ftv}(\theta_1) \cap New' = \emptyset.\end{aligned} \tag{49}$$

and

$$\begin{aligned}&R_2(S_2S_1\theta_2)\\ &= P\rho && \text{by (49) and the definition of } G_1\\ &= PG_1\rho && \text{because } \mathit{ftv}(\rho) \cap \mathit{supp}(G_1) = \emptyset\\ &= PG_1G_2\rho && \text{because } \mathit{ftv}(\rho) \cap \mathit{supp}(G_2) = \emptyset\\ &= P'\rho && \text{because the new } \beta_1, \beta_2 \notin \mathit{ftv}(\rho)\\ &= R_2(S_2S_1\rho) && \text{by (48) and because } \mathit{ftv}(\rho) \cap New' = \emptyset.\end{aligned}$$

Thus the unification at ($\mathcal{G}$.16) succeeds with S_3 such that for a substitution R_3,

$$R_3S_3 = R_2. \tag{50}$$

Hence $\mathcal{G}(\Gamma, \texttt{fix}\ f\ \lambda x.e, \rho)$ succeeds with $S_3S_2S_1$, and $(R_3S_3S_2S_1)\restriction_{New} = P\restriction_{New}$ because

$$\begin{array}{ll}(R_3S_3S_2S_1)|_{New} & \\ =(R_2S_2S_1)|_{New} & \text{by (50)} \\ =P'|_{New} & \text{by (48)} \\ =P|_{New} & \text{because } \mathit{supp}(G_1)\cup\mathit{supp}(G_2)\cup\{\beta_1,\beta_2\}\subseteq \mathit{New}.\end{array}$$

■

§A3 Proof of Lemma A2.4

We prove by structural induction on e.

- **case** $()$: By Theorem 1.1, $\mathit{itv}(\mathcal{U}(\rho,\iota)) \subseteq \mathit{ftv}(\rho) \cup \mathit{ftv}(\iota) = \mathit{ftv}(\rho)$.
- **case** x :
$$\begin{array}{ll}\mathit{itv}(\mathcal{U}(\rho,\{\vec{\beta}/\vec{\alpha}\}\tau)) & \\ \subseteq \mathit{ftv}(\rho)\cup\mathit{ftv}(\{\vec{\beta}/\vec{\alpha}\}\tau) & \text{by Theorem 1.1} \\ \subseteq \mathit{ftv}(\rho)\cup(\mathit{ftv}(\tau)\setminus\vec{\alpha})\cup\vec{\beta} & \\ = \mathit{ftv}(\rho)\cup\mathit{ftv}(\forall\vec{\alpha}.\tau)\cup\vec{\beta} & \\ = \mathit{ftv}(\rho)\cup\mathit{ftv}(\Gamma(x))\cup\vec{\beta} & \text{because } \Gamma(x)=\forall\vec{\alpha}.\tau \\ \subseteq \mathit{ftv}(\rho)\cup\mathit{ftv}(\Gamma)\cup\vec{\beta}. & \end{array}$$
Note that $\vec{\beta}$ is the set of new type variables used by $\mathcal{G}(\Gamma,x,\rho)$.
- **case** $\lambda x.e$: Let G be the substitution for $\theta \geq \rho$ at $(\mathcal{G}.3)$. Note that all the type variables in $\mathit{supp}(G)$ are new by definition.
$$\begin{array}{ll}\mathit{itv}(S_1) & \\ \subseteq \mathit{ftv}(\theta)\cup\mathit{ftv}(\beta_1\to\beta_2) & \text{by Theorem 1.1} \\ \subseteq \mathit{ftv}(\rho)\cup\mathit{supp}(G)\cup\{\beta_1,\beta_2\} & \text{because } \mathit{supp}(G)=\mathit{ftv}(\theta)\setminus\mathit{ftv}(\rho), \\ \mathit{itv}(S_2) & \\ \subseteq \mathit{ftv}(S_1\Gamma)\cup\mathit{ftv}(S_1\beta_1)\cup\mathit{ftv}(S_1\beta_2)\cup \mathit{New}_1 & \text{by induction} \\ \subseteq \mathit{itv}(S_1)\cup\mathit{ftv}(\Gamma)\cup\{\beta_1,\beta_2\}\cup \mathit{New}_1 & \text{by Lemma A2.3}\end{array}$$
where New_1 is the set of new type variables used by $\mathcal{G}(S_1\Gamma{+}x{:}\ S_1\beta_1, e, S_1\beta_2)$ at $(\mathcal{G}.4)$, and
$$\begin{array}{ll}\mathit{itv}(S_3) & \\ \subseteq \mathit{ftv}(S_2S_1\theta)\cup\mathit{ftv}(S_2S_1\rho) & \text{by Theorem 1.1} \\ \subseteq \mathit{itv}(S_2)\cup\mathit{itv}(S_1)\cup\mathit{ftv}(\theta)\cup\mathit{ftv}(\rho) & \text{by Lemma A2.3} \\ \subseteq \mathit{itv}(S_2)\cup\mathit{itv}(S_1)\cup\mathit{supp}(G)\cup\mathit{ftv}(\rho). & \end{array}$$
Therefore $\mathit{itv}(S_3S_2S_1) \subseteq \mathit{ftv}(\Gamma)\cup\mathit{ftv}(\rho)\cup(\mathit{supp}(G)\cup\{\beta_1,\beta_2\}\cup \mathit{New}_1)$. Note that $\mathit{supp}(G)\cup\{\beta_1,\beta_2\}\cup \mathit{New}_1$ is the set of new type variables used by $\mathcal{G}(\Gamma,\lambda x.e,\rho)$.

Other cases can be similarly proven. ■

§A4 Relative Earliness Proof

Theorem 2.3

Let A and A' be instances of $\mathcal{G}$ such that $A \sqsubseteq A'$, Γ_0 be a type environment, e_0 be an expression, and ρ_0 be a type. If $[\![A(\Gamma_0,e_0,\rho_0)]\!]$ has $(\Gamma,e,\rho)^{d/u}$, then $[\![A'(\Gamma_0,e_0,\rho_0)]\!]$ has $(\Gamma',e,\rho')^{d/u}$ and there exists a substitution R such that $R\Gamma' \succ \Gamma$ and $R\rho' = \rho$. The theorem also holds for $\mathcal{G}^R$.

The proof of Theorem 2.3 uses Lemmas A4.1 and A4.2.

Lemma A4.1 (Lee and Yi[10])
If $\Gamma \succ \Gamma'$ then $Clos_{\Gamma}(\tau) \succ Clos_{\Gamma'}(\tau)$.

Lemma A4.2
Let A and A' be instances of $\mathcal{G}$, Γ and Γ' be type environments, and ρ and ρ' be types such that $R\Gamma' \succ \Gamma$ and $R\rho' = \rho$ 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 $(R'S')|_{New} = (SR)|_{New}$ where *New* is the set of new type variables used by $A'(\Gamma', e, \rho')$. The lemma also holds for $\mathcal{G}^R$.

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

$$S\Gamma \vdash e : S\rho.$$

By Lemma A1.2, $SR\Gamma' \succ S\Gamma$ and $S\rho = SR\rho'$. Thus by Lemma A2.2,

$$SR\Gamma' \vdash e : SR\rho'.$$

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 2.3
We prove by induction on the length of the prefixes of $[\![A(\Gamma_0, e_0, \rho_0)]\!]$, and we prove for $\mathcal{G}$ and $\mathcal{G}^R$ simultaneously. 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: $(\Gamma_0, e_0, \rho_0)^d \cdots (\Gamma, e, \rho)^u$.

- **case of the return from** e: The case means that $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has
  $$(\Gamma, e, \rho)^d \cdots (\Gamma, e, \rho)^u.$$
  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 A4.2, $A'(\Gamma', e, \rho')$ succeeds; that is, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', e, \rho')^u$.

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

- **case** e **in** $\lambda x.e$: that is, $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has
  $$(\Gamma, \lambda x.e, \rho)^d (S_1\Gamma + x\colon S_1\beta_1, e, S_1\beta_2)^d$$
  where $S_1 = \mathcal{U}(\beta_1 \to \beta_2, \theta)$ at ($\mathcal{G}$.3), and β_1 and β_2 are the new type variables at ($\mathcal{G}$.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 $R\rho' = \rho$ and
  $$R\Gamma' \succ \Gamma. \qquad (51)$$

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

$$\theta = (R|_{supp(P)} \cup P)\theta' \tag{52}$$

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

$$supp(P) \cap ftv(\Gamma') = \emptyset. \tag{53}$$

Let $R_0 = R|_{\{\beta_1', \beta_2'\} \cup supp(P)} \cup P \cup \{\beta_1/\beta_1', \beta_2/\beta_2'\}$ where β_1' and β_2' are the new type variables of A' introduced at $(\mathcal{G}.3)$. Then $S_1 R_0$ unifies $\beta_1' \to \beta_2'$ and θ' at $(\mathcal{G}.3)$ because

$$\begin{array}{ll} S_1 R_0(\theta') & \\ = S_1(R|_{supp(P)} \cup P)\theta' & \text{because the new } \beta_1', \beta_2' \notin ftv(\theta') \\ = S_1 \theta & \text{by (52)} \\ = S_1(\beta_1 \to \beta_2) & \text{by } (\mathcal{G}.3) \\ = S_1 R_0(\beta_1' \to \beta_2') & \text{by the definition of } R_0. \end{array}$$

Thus the unification of A' at $(\mathcal{G}.3)$ succeeds with S_1', hence $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(S_1'\Gamma' + x\colon S_1'\beta_1', e, S_1'\beta_2')^d$.

Now we prove the rest that there exists a substitution R' such that $R'(S_1'\Gamma' + x\colon S_1'\beta_1') \succ (S_1\Gamma + x\colon S_1\beta_1)$ and $R'(S_1'\beta_2') = S_1\beta_2$. Because $(\mathcal{G}.3)$ succeeds with S_1', by Theorem 1.1, there exists a substitution R_1 such that

$$S_1 R_0 = R_1 S_1'. \tag{54}$$

Then such R' is R_1 because

$$\begin{array}{ll} R_1(S_1'\Gamma' + x\colon S_1'\beta_1') & \\ = S_1 R_0(\Gamma' + x\colon \beta_1') & \text{by (54)} \\ = S_1((R|_{supp(P)} \cup P)\Gamma' + x\colon R_0\beta_1') & \\ & \text{because the new } \beta_1', \beta_2' \notin ftv(\Gamma') \\ = S_1(R\Gamma' + x\colon \beta_1) & \text{by (53) and the definition of } R_0 \\ \succ S_1(\Gamma + x\colon \beta_1) & \text{by (51) and Lemma A1.2} \end{array}$$

and

$$\begin{array}{lcll} R_1(S_1'\beta_2') & = & S_1 R_0 \beta_2' & \text{by (54)} \\ & = & S_1\beta_2 & \text{by the definition of } R_0. \end{array}$$

- **case e in $e\ e_2$ for instances of $\mathcal{G}$:** that is, $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has

$$(\Gamma, e\ e_2, \rho)^d (\Gamma, e, \theta_1)^d$$

where θ_1 is the type relaxed from $\beta \to \rho$ at $(\mathcal{G}.8)$. 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 $R\rho' = \rho$ and

$$R\Gamma' \succ \Gamma. \tag{55}$$

Thus by the definition of $\mathcal{G}$, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', e, \theta_1')^d$ where θ_1' is the type relaxed from $\beta' \to \rho'$ at $(\mathcal{G}.8)$.

Now we prove the rest. Let $R_0 = R|_{\{\beta'\}} \cup \{\beta/\beta'\}$ where β and β' are respectively the new type variables of A and A' at $(\mathcal{G}.6)$. Because $A \sqsubseteq A'$ and

$$\begin{array}{lcll} R_0(\beta' \to \rho') & = & \beta \to R\rho' & \text{because the new } \beta' \notin ftv(\rho') \\ & = & \beta \to \rho, & \end{array}$$

there exists a substitution P such that

$$(R_0\!\restriction_{supp(P)} \cup P)\theta'_1 = \theta_1$$

and $supp(P) \subseteq ftv(\theta'_1) \setminus ftv(\beta' \to \rho')$. Note that $supp(P) \cap ftv(\Gamma') = \emptyset$ by the definition of $\mathcal{G}$. Thus

$$\begin{array}{ll} (R_0\!\restriction_{supp(P)} \cup P)(\Gamma') & \\ = R\Gamma' & \text{because } (\{\beta\} \cup supp(P)) \cap ftv(\Gamma') = \emptyset \\ \succ \Gamma & \text{by (55).} \end{array}$$

- **case e in $e_1\ e$ for instances of $\mathcal{G}$:** that is, $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has

  $$(\Gamma, e_1\ e, \rho)^d (\Gamma, e_1, \theta_1)^d \cdots (\Gamma, e_1, \theta_1)^u (S_2 S_1 \Gamma, e, \theta_3)^d$$

  where θ_1, θ_2, and θ_3 are respectively the relaxed types of A at ($\mathcal{G}$.6), ($\mathcal{G}$.7), and ($\mathcal{G}$.8), $S_1 = \mathcal{G}(\Gamma, e_1, \theta_1)$ at ($\mathcal{G}$.6), and $S_2 = \mathcal{U}(S_1\theta_1, \theta_2)$ at ($\mathcal{G}$.7).
  By induction hypothesis, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', e_1\ e, \rho')^d$ and there exists a substitution R such that $R\rho' = \rho$ and

  $$R\Gamma' \succ \Gamma. \tag{56}$$

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

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

    $$\begin{array}{rcll} R_0(\beta' \to \rho') & = & \beta \to R\rho' & \text{because the new } \beta' \notin ftv(\rho') \\ & = & \beta \to \rho, & \end{array} \tag{57}$$

    there exists a substitution P_1 such that

    $$\theta_1 = (R_0\!\restriction_{supp(P_1)} \cup P_1)\theta'_1 \tag{58}$$

    and $supp(P_1) \subseteq ftv(\theta'_1) \setminus ftv(\beta' \to \rho')$. Note that by the definition of $\mathcal{G}$,

    $$supp(P_1) \cap (ftv(\Gamma') \cup ftv(\beta' \to \rho')) = \emptyset \tag{59}$$

    and thus

    $$\begin{array}{rcll} (R_0\!\restriction_{supp(P_1)} \cup P_1)\Gamma' & = & R\Gamma' & \text{by (59) and } \beta' \notin ftv(\Gamma') \\ & \succ & \Gamma & \text{by (56).} \end{array} \tag{60}$$

    Because $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma, e_1, \theta_1)^u$, $(R_0\!\restriction_{supp(P_1)} \cup P_1)\Gamma' \succ \Gamma$ (60), and $(R_0\!\restriction_{supp(P_1)} \cup P_1)\theta'_1 = \theta_1$ (58), by Lemma A4.2, $A'(\Gamma', e_1, \theta'_1)$ succeeds with S'_1 such that for a substitution R_1,

    $$(R_1 S'_1)\!\restriction_{New_1} = (S_1(R_0\!\restriction_{supp(P_1)} \cup P_1))\!\restriction_{New_1} \tag{61}$$

    where New_1 is the set of new type variables used by $A'(\Gamma', e_1, \theta'_1)$.
  - $\mathcal{U}(S'_1\theta'_1, \theta'_2)$ at ($\mathcal{G}$.7) succeeds: Because $A \sqsubseteq A'$ and

    $$\begin{array}{ll} R_1(S'_1(\beta' \to \rho')) & \\ = S_1(R_0\!\restriction_{supp(P_1)} \cup P_1)(\beta' \to \rho') & \\ \qquad \text{by (61) and because } ftv(\beta' \to \rho') \cap New_1 = \emptyset & \\ = S_1 R_0(\beta' \to \rho') & \text{by (59)} \\ = S_1(\beta \to \rho) & \text{by (57),} \end{array} \tag{62}$$

there exists a substitution P_2 such that

$$\theta_2 = (R_1|_{supp(P_2)} \cup P_2)\theta_2' \tag{63}$$

and $supp(P_2) \subseteq ftv(\theta_2') \setminus ftv(S_1'(\beta' \to \rho'))$. Note that

$$\begin{array}{ll} ftv(S_1'\theta_1') \cup ftv(S_1'\beta') \cup ftv(S_1'\Gamma') & \\ \subseteq itv(S_1') \cup ftv(\theta_1') \cup \{\beta'\} \cup ftv(\Gamma') & \text{by Lemma A2.3} \\ \subseteq New_1 \cup ftv(\theta_1') \cup \{\beta'\} \cup ftv(\Gamma') & \text{by Lemma A2.4} \end{array}$$

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

$$supp(P_2) \cap (ftv(S_1'\theta_1') \cup ftv(S_1'\beta') \cup ftv(S_1'\Gamma')) = \emptyset \tag{64}$$

Then $S_2(R_1|_{supp(P_2)} \cup P_2)$ unifies $S_1'\theta_1'$ and θ_2' at $(\mathcal{G}.7)$ because

$$\begin{array}{ll} S_2(R_1|_{supp(P_2)} \cup P_2)(S_1'\theta_1') & \\ = S_2R_1S_1'\theta_1' & \text{by (64)} \\ = S_2S_1(R_0|_{supp(P_1)} \cup P_1)\theta_1' & \text{by (61) and} \\ & \text{because } ftv(\theta_1') \cap New_1 = \emptyset \\ = S_2S_1\theta_1 & \text{by (58)} \\ = S_2\theta_2 & \text{by } (\mathcal{G}.7) \\ = S_2(R_1|_{supp(P_2)} \cup P_2)(\theta_2') & \text{by (63).} \end{array}$$

Thus the unification of A' at $(\mathcal{G}.7)$ succeeds with S_2'.

Therefore $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(S_2'S_1'\Gamma', e, \theta_3')^d$.
Now we prove the rest that there exists a substitution R' such that $R'\theta_3' = \theta_3$ and $R'(S_2'S_1'\Gamma') \succ S_2S_1\Gamma$. Because $(\mathcal{G}.7)$ succeeds with S_2', by Theorem 1.1, there exists a substitution R_2 such that

$$R_2S_2' = S_2(R_1|_{supp(P_2)} \cup P_2). \tag{65}$$

Because $A \sqsubseteq A'$ and

$$\begin{array}{rcll} R_2(S_2'S_1'\beta') & = & S_2(R_1|_{supp(P_2)} \cup P_2)S_1'\beta' & \text{by (65)} \\ & = & S_2R_1S_1'\beta' & \text{by (64)} \\ & = & S_2S_1\beta & \text{by (62),} \end{array}$$

there exists a substitution P_3 such that

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

and $supp(P_3) \subseteq ftv(\theta_3') \setminus ftv(S_2'S_1'\beta')$. Note again that, by Lemma A2.3 and A2.4 and Theorem 1.1,

$$\begin{array}{l} ftv(S_2'S_1'\Gamma') \\ \subseteq ftv(\theta_1) \cup ftv(\theta_2) \cup New_1 \cup ftv(\Gamma') \\ \subseteq supp(P_1) \cup ftv(\beta \to \rho) \cup supp(P_2) \cup New_1 \cup ftv(\Gamma') \end{array}$$

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

$$supp(P_3) \cap ftv(S_2'S_1'\Gamma') = \emptyset. \tag{66}$$

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

$$\begin{array}{ll}
(R_2\restriction_{supp(P_3)} \cup P_3)(S_2'S_1'\Gamma') & \\
= R_2 S_2' S_1' \Gamma' & \text{by (66)} \\
= S_2(R_1\restriction_{supp(P_2)} \cup P_2)S_1'\Gamma' & \text{by (65)} \\
= S_2 R_1 S_1' \Gamma' & \text{by (64)} \\
= S_2 S_1 (R_0\restriction_{supp(P_1)} \cup P_1)\Gamma' & \text{by (61) and because} \\
 & ftv(\Gamma') \cap New_1 = \emptyset \\
\succ S_2 S_1 \Gamma & \text{by (60) and Lemma A1.2.}
\end{array}$$

- **case e in $e_1\ e$ for instances of $\mathcal{G}^R$:** that is, $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has

$$(\Gamma, e_1\ e, \rho)^d (\Gamma, e, \beta)^d$$

where β is the new type variable introduced at ($\mathcal{G}$.18). By induction, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', e_1\ e, \rho')^d$ and there exists a substitution R such that $R\Gamma' \succ \Gamma$ and $R\rho' = \rho$. By the definition of $\mathcal{G}^R$, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', e, \beta')^d$ where β' is the new type variable introduced at ($\mathcal{G}$.18). Let $R_0 = R\restriction_{\{\beta'\}} \cup \{\beta/\beta'\}$. Then $R_0\Gamma' = R\Gamma' \succ \Gamma$ and $R_0\beta' = \beta$.

- **case e in $e\ e_2$ for instances of $\mathcal{G}^R$:** that is, $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has

$$(\Gamma, e\ e_2, \rho)^d (\Gamma, e_2, \beta)^d \cdots (\Gamma, e_2, \beta)^u (S_1\Gamma, e, \theta)^d$$

where β is the new type variable introduced at ($\mathcal{G}$.18), θ is the relaxed type at ($\mathcal{G}$.19), and $S_1 = \mathcal{G}^R(\Gamma, e_2, \beta)$ at ($\mathcal{G}$.18).
By induction, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', e\ e_2, \rho')^d$ and there exists a substitution R such that $R\rho' = \rho$ and

$$R\Gamma' \succ \Gamma. \tag{67}$$

Let $R_0 = R\restriction_{\{\beta'\}} \cup \{\beta/\beta'\}$ where β' is the new type variable introduced at ($\mathcal{G}$.18). Then $R_0\beta' = \beta$ and

$$\begin{array}{rll}
R_0\Gamma' & = R\Gamma' & \text{because the new } \beta' \notin ftv(\Gamma') \\
 & \succ \Gamma & \text{by (67).}
\end{array} \tag{68}$$

Thus by Lemma A4.2, $A'(\Gamma', e_2, \beta')$ at ($\mathcal{G}$.18) succeeds with S_1', hence $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(S_1'\Gamma', e_1, \theta')^d$.
Now we prove the rest that there exists a substitution R' such that $R'\theta' = \theta$ and $R'S_1'\Gamma' \succ S_1\Gamma$. Because ($\mathcal{G}$.18) succeeds with S_1', by Lemma A4.2, there is a substitution R_1 such that

$$(R_1 S_1')\restriction_{New_1} = (S_1 R_0)\restriction_{New_1} \tag{69}$$

where New_1 is the set of new type variables used by $A'(\Gamma', e_2, \beta')$.
Because $A \sqsubseteq A'$ and

$$\begin{array}{ll}
R_1(S_1'\rho') & \\
= S_1 R_0 \rho' & \text{by (69) and because } ftv(\rho') \cap New_1 = \emptyset \\
= S_1 R \rho' & \text{because the new } \beta' \notin ftv(\rho') \\
= S_1 \rho, &
\end{array}$$

there exists a substitution P such that

$$(R_1\restriction_{supp(P)} \cup P)\theta' = \theta$$

and $supp(P) \subseteq ftv(\theta') \setminus ftv(S_1'\rho')$. Note that

$$supp(P) \cap (ftv(S_1'\Gamma') \cup ftv(S_1'\beta')) = \emptyset. \tag{70}$$

by the definition of $\mathcal{G}$ because

$$\begin{aligned}&ftv(S_1'\Gamma') \cup ftv(S_1'\beta')\\ &\subseteq itv(S_1') \cup ftv(\Gamma') \cup \{\beta'\} && \text{by Lemma A2.3}\\ &\subseteq New_1 \cup ftv(\Gamma') \cup \{\beta'\} && \text{by Lemma A2.4.}\end{aligned}$$

Therefore, such R' is $(R_1\downharpoonright_{supp(P)} \cup P)$ because

$$\begin{aligned}&(R_1\downharpoonright_{supp(P)} \cup P)(S_1'\Gamma')\\ &= R_1(S_1'\Gamma') && \text{by (70)}\\ &= S_1R_0\Gamma' && \text{by (69) and because } New_1 \cap ftv(\Gamma') = \emptyset\\ &\succ S_1\Gamma && \text{by (68) and Lemma A1.2.}\end{aligned}$$

- **case** e **in** (`let` x=e `in` e_2): that is, $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has

  $$(\Gamma, \texttt{let } x{=}e \texttt{ in } e_2, \rho)^d(\Gamma, e, \beta)^d$$

  where β is the new type variable introduced at ($\mathcal{G}$.11). By induction, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', \texttt{let } x{=}e \texttt{ in } e_2, \rho')^d$ and there exists a substitution R such that $R\Gamma' \succ \Gamma$ and $R\rho' = \rho$. By the definition of $\mathcal{G}$, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', e_1, \beta')^d$ where β' is the new type variable introduced at ($\mathcal{G}$.11). Let $R_0 = R\downharpoonright_{\{\beta'\}} \cup \{\beta/\beta'\}$. Then $R_0\Gamma' = R\Gamma' \succ \Gamma$ and $R_0\beta' = \beta$.
- **case** e **in** (`let` x=e_1 `in` e): that is, $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has

  $$(\Gamma, \texttt{let } x{=}e_1 \texttt{ in } e, \rho)^d(\Gamma, e, \beta)^d \cdots (\Gamma, e, \beta)^u(S_1\Gamma + x\colon Clos_{S_1\Gamma}(S_1\beta), e, \theta)^d$$

  where β is the new type variable introduced at ($\mathcal{G}$.11), θ is the relaxed type at ($\mathcal{G}$.12), and $S_1 = \mathcal{G}(\Gamma, e_1, \beta)$ at ($\mathcal{G}$.11).
  By induction, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', \texttt{let } x{=}e \texttt{ in } e_2, \rho')^d$ and there exists a substitution R such that $R\rho' = \rho$ and

  $$R\Gamma' \succ \Gamma. \tag{71}$$

  Let $R_0 = R\downharpoonright_{\{\beta'\}} \cup \{\beta/\beta'\}$ where β' is the new type variable introduced at ($\mathcal{G}$.11). Then $R_0\beta' = \beta$ and

  $$\begin{aligned}R_0\Gamma' &= R\Gamma' && \text{because the new } \beta' \notin ftv(\Gamma')\\ &\succ \Gamma && \text{by (71).}\end{aligned} \tag{72}$$

  Thus by Lemma A4.2, $A'(\Gamma', e_1, \beta')$ at ($\mathcal{G}$.11) succeeds with S_1', hence $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(S_1'\Gamma' + x\colon Clos_{S_1'\Gamma'}(S_1'\beta'), e_2, \theta')^d$.
  Now we prove the rest that there exists a substitution R' such that $R'\theta' = \theta$ and $R'(S_1'\Gamma' + x\colon Clos_{S_1'\Gamma'}(S_1'\beta')) \succ S_1\Gamma + x\colon Clos_{S_1\Gamma}(S_1\beta)$. Because ($\mathcal{G}$.11) succeeds with S_1', by Lemma A4.2, there is a substitution R_1 such that

  $$(R_1S_1')\downharpoonright_{New_1} = (S_1R_0)\downharpoonright_{New_1} \tag{73}$$

  where New_1 is the set of new type variables used by $A'(\Gamma', e_1, \beta')$.
  Because $A \sqsubseteq A'$ and

  $$\begin{aligned}&R_1(S_1'\rho')\\ &= S_1R_0\rho' && \text{by (73) and because } ftv(\rho') \cap New_1 = \emptyset\\ &= S_1R\rho' && \text{because the new } \beta' \notin ftv(\rho')\\ &= S_1\rho,\end{aligned}$$

  there exists a substitution P such that

  $$(R_1\downharpoonright_{supp(P)} \cup P)\theta' = \theta$$

and $supp(P) \subseteq ftv(\theta') \setminus ftv(S_1'\rho')$. Note that

$$\begin{array}{ll} ftv(S_1'\Gamma') \cup ftv(S_1'\beta') & \\ \subseteq itv(S_1') \cup ftv(\Gamma') \cup \{\beta'\} & \text{by Lemma A2.3} \\ \subseteq New_1 \cup ftv(\Gamma') \cup \{\beta'\} & \text{by Lemma A2.4} \end{array}$$

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

$$supp(P) \cap (ftv(S_1'\Gamma') \cup ftv(S_1'\beta')) = \emptyset. \tag{74}$$

Therefore, such R' is $(R_1|_{\overline{supp(P)}} \cup P)$ because

$$\begin{array}{ll} (R_1|_{\overline{supp(P)}} \cup P)(S_1'\Gamma') & \\ = R_1(S_1'\Gamma') & \text{by (74)} \\ = S_1R_0\Gamma' & \text{by (73) and because } New_1 \cap ftv(\Gamma') = \emptyset \\ \succ S_1\Gamma & \text{by (72) and Lemma A1.2} \end{array} \tag{75}$$

and

$$\begin{array}{ll} (R_1|_{\overline{supp(P)}} \cup P)(Clos_{S_1'\Gamma'}(S_1'\beta')) & \\ = R_1 Clos_{S_1'\Gamma'}(S_1'\beta') & \text{by (74)} \\ \succ Clos_{R_1S_1'\Gamma'}(R_1S_1'\beta') & \text{by Lemma A2.1} \\ \succ Clos_{S_1\Gamma}(R_1S_1'\beta') & \text{by (75) and Lemma A4.1} \\ = Clos_{S_1\Gamma}(S_1R_0\beta') & \text{by (73) and } \beta' \notin New_1 \\ = Clos_{S_1\Gamma}(S_1\beta) & \text{by the definition of } R_0. \end{array}$$

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

$$(\Gamma, \texttt{fix}\ f\ \lambda x.e, \rho)^d (S_1\Gamma_1 + x\colon S_1\beta_1, e, S_1\beta_2)^d$$

where $\Gamma_1 = \Gamma + f : \theta_1$ at ($\mathcal{G}$.14), $S_1 = \mathcal{U}(\beta_1 \to \beta_2, \theta)$ at ($\mathcal{G}$.15), and β_1 and β_2 are the new type variables at ($\mathcal{G}$.15). By induction, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', \texttt{fix}\ f\ \lambda x.e, \rho')^d$ and there exists a substitution R such that $R\rho' = \rho$ and

$$R\Gamma' \succ \Gamma. \tag{76}$$

In order for $A'(\Gamma', \texttt{fix}\ f\ \lambda x.e, \rho')$ to have a call for e, the unification at ($\mathcal{G}$.15) must hold. Because $A \sqsubseteq A'$, there exists a substitution P such that

$$\theta_1 = (R|_{\overline{supp(P)}} \cup P)\theta_1', \tag{77}$$

$$\theta_2 = (R|_{\overline{supp(P)}} \cup P)\theta_2', \tag{78}$$

and $supp(P) \subseteq (ftv(\theta_1) \cup ftv(\theta_2)) \setminus ftv(\rho')$. Note that by the definition of $\mathcal{G}$,

$$supp(P) \cap ftv(\Gamma') = \emptyset. \tag{79}$$

Let $R_0 = R|_{\overline{\{\beta_1', \beta_2'\} \cup supp(P)}} \cup P \cup \{\beta_1/\beta_1', \beta_2/\beta_2'\}$ where β_1' and β_2' are the new type variables of A' introduced at ($\mathcal{G}$.15). Then S_1R_0 unifies $\beta_1' \to \beta_2'$ and θ_2' at ($\mathcal{G}$.15) because

$$\begin{array}{ll} S_1R_0(\theta_2') & \\ = S_1(R|_{\overline{supp(P)}} \cup P)\theta_2' & \text{because the new } \beta_1', \beta_2' \notin ftv(\theta_2') \\ = S_1\theta_2 & \text{by (78)} \\ = S_1(\beta_1 \to \beta_2) & \text{by } (\mathcal{G}.15) \\ = S_1R_0(\beta_1' \to \beta_2') & \text{by the definition of } R_0. \end{array}$$

Thus the unification of A' at ($\mathcal{G}$.15) succeeds with S_1', hence $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(S_1'\Gamma_1' + x\colon S_1'\beta_1', e, S_1'\beta_2')^d$.

Now we prove the rest that there exists a substitution R' such that $R'(S_1'\Gamma_1'+x\colon S_1'\beta_1') \succ (S_1\Gamma_1+x\colon S_1\beta_1)$ and $R'(S_1'\beta_2') = S_1\beta_2$. Because $(\mathcal{G}.15)$ succeeds with S_1', by Theorem 1.1, there exists a substitution R_1 such that

$$S_1R_0 = R_1S_1'. \qquad (80)$$

Then such R' is R_1 because

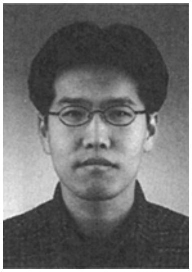
$$\begin{array}{ll}
R_1(S_1'\Gamma_1'+x\colon S_1'\beta_1') & \\
= S_1R_0(\Gamma_1'+x\colon \beta_1') & \text{by (80)} \\
= S_1((R|_{supp(P)} \cup P)\Gamma_1'+x\colon R_0\beta_1') & \\
 & \text{because the new } \beta_1', \beta_2' \not\in ftv(\Gamma_1') \\
= S_1((R|_{supp(P)} \cup P)\Gamma_1'+x\colon \beta_1) & \text{by the definition of } R_0 \\
= S_1((R|_{supp(P)} \cup P)(\Gamma'+f\colon \theta_1')+x\colon \beta_1) & \text{by } (\mathcal{G}.14) \\
= S_1(R\Gamma'+f\colon \theta_1+x\colon \beta_1) & \text{by (77) and (79)} \\
\succ S_1(\Gamma+f\colon \theta_1+x\colon \beta_1) & \text{by (76) and Lemma A1.2} \\
= S_1(\Gamma_1+x\colon \beta_1) & \text{by } (\mathcal{G}.14)
\end{array}$$

and

$$\begin{array}{rcll}
R_1(S_1'\beta_2') & = & S_1R_0\beta_2' & \text{by (80)} \\
 & = & S_1\beta_2 & \text{by the definition of } R_0.
\end{array}$$

■

Hyunjun Eo: He is a Ph.D. candidate in the Department of Computer Science at KAIST (Korea Advanced Institute of Science and Technology). He recieved his bachelor's degree and master's degree in Computer Science from KAIST in 1996 and 1998, respectively. His research interest has been on static program analysis, fixpoint iteration algorithm and higher-order and typed languages. From fall 1998, he has been a research assistant of the National Creative Research Initiative Center for Research on Program Analysis System. He is currently working on developing a tool for automatic generation of program analyzer.



Oukseh Lee: He is a Ph.D. candidate in the Department of Computer Science at KAIST (Korea Advanced Institute of Science and Technology). He received his bachelor's and master's degree in Computer Science from KAIST in 1995 and 1997, respectively. His research interest has been on static program analysis, type system, program language implementation, higher-order and typed languages, and program verification. From 1998, he has been a research assistant of the National Creative Research Initiative Center for Research on Program Analysis System. He is currently working on compile-time analyses and verification for the memory behavior of programs.

Kwangkeun Yi, Ph.D.: His research interest has been on semantic-based program analysis and systems application of language technologies. After his Ph.D. from University of Illinois at Urbana-Champaign he joined the Software Principles Research Department at Bell Laboratories, where he worked on various static analysis approaches for higher-order and typed programming languages. For 1995 to 2003 he was a faculty member in the Department of Computer Science, Korea Advanced Institute of Science and Technology. Since fall 2003, he has been a faculty member in the School of Computer Science and Engineering, Seoul National University.