ROPAS
Research On Program Analysis System
National Creative Research Initiative Center
Korea Advanced Institute of Science and Technology

ROPAS MEMO
2000-5
March 31, 2000

A Generalized Let-Polymorphic Type Inference Algorithm

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

Abstract

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

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

1 Introduction

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

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

[0]This work is supported by Creative Research Initiatives of the Korean Ministry of Science and Technology.

We present a generalized let-polymorphic type inference algorithm, prove that any of its instances is sound and complete with respect to the Hindley/Milner let-polymorphic type system, and find a condition on two instance algorithms so that one algorithm should find type errors earlier than the other. By instantiating the generalized algorithm with different parameters, we can achieve not only the two opposite algorithms ($\mathcal{W}$ and $\mathcal{M}$) but also other various hybrid algorithms that avoid their extremities in type-checking. Such hybrid algorithms' soundness and completeness follow automatically, and their relative earliness in detecting type-errors can be decided by checking a simple condition. The set of hybrid algorithms that come from the generalized algorithm is a superset of the existing hybrid algorithms in SML/NJ[sml99] and OCaml[LRVD99].

1.1 Notation

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

2 The Generalized Algorithm $\mathcal{G}$

2.1 Overview

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

Our generalized algorithm is based on the top-down, context-sensitive algorithm $\mathcal{M}$. Key observation is that varying the type-checking strategy is possible by changing two factors in $\mathcal{M}$: the information amount of the type constraints to pass to type-checking sub-expressions and the places of the unification. Algorithm $\mathcal{M}$ carries as much information as possible at 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. By tuning the two factors, other type-checking strategies are also possible:

Example 1 Consider an application expression

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

Abstract Syntax

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

(CON) $$\Gamma \vdash \texttt{()} : \iota$$

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

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

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

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

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

Figure 1: Language and its let-polymorphic type system

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 achieve 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 enforces that `IsOne`'s type be just a function, whatever its domain and range types are. After its success, we check whether the function's range type is `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`.

$$\mathcal{W}\colon \mathit{TypEnv} \times \mathit{Expr} \to \mathit{Subst} \times \mathit{Type}$$

$$
\begin{array}{llll}
\mathcal{W}(\Gamma,\, \texttt{()}) & = & (id,\, \iota) & (\mathcal{W}.1)\\
\mathcal{W}(\Gamma,\, x) & = & (id,\, \{\vec{\beta}/\vec{\alpha}\}\tau) \ \text{ where } \Gamma(x) = \forall\vec{\alpha}.\tau,\ \text{new } \vec{\beta} & (\mathcal{W}.2)\\
\mathcal{W}(\Gamma, \lambda x.e) & = & \text{let} \ \ (S_1,\, \tau_1) = \mathcal{W}(\Gamma + x\colon\beta,\, e),\ \text{new } \beta & (\mathcal{W}.3)\\
 & & \text{in} \ \ (S_1,\, S_1\beta \to \tau_1) & \\
\mathcal{W}(\Gamma,\, e_1\ e_2) & = & \text{let} \ \ (S_1,\, \tau_1) = \mathcal{W}(\Gamma,\, e_1) & (\mathcal{W}.4)\\
 & & \phantom{\text{let}} \ \ (S_2,\, \tau_2) = \mathcal{W}(S_1\Gamma,\, e_2) & (\mathcal{W}.5)\\
 & & \phantom{\text{let}} \ \ S_3 = \mathcal{U}(S_2\tau_1,\, \tau_2 \to \beta),\ \text{new } \beta & (\mathcal{W}.6)\\
 & & \text{in} \ \ (S_3S_2S_1,\, S_3\beta) & \\
\multicolumn{3}{l}{\mathcal{W}(\Gamma,\, \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2) =} & \\
 & & \text{let} \ \ (S_1,\, \tau_1) = \mathcal{W}(\Gamma,\, e_1) & (\mathcal{W}.7)\\
 & & \phantom{\text{let}} \ \ (S_2,\, \tau_2) = \mathcal{W}(S_1\Gamma + x\colon \mathit{Clos}_{S_1\Gamma}(\tau_1),\, e_2) & (\mathcal{W}.8)\\
 & & \text{in} \ \ (S_2S_1,\, \tau_2) & \\
\mathcal{W}(\Gamma,\, \texttt{fix}\ f\ \lambda x.e) & = & \text{let} \ \ (S_1,\, \tau_1) = \mathcal{W}(\Gamma + f\colon\beta,\, \lambda x.e),\ \text{new } \beta & (\mathcal{W}.9)\\
 & & \phantom{\text{let}} \ \ S_2 = \mathcal{U}(S_1\beta,\, \tau_1) & (\mathcal{W}.10)\\
 & & \text{in} \ \ (S_2S_1,\, S_2\tau_1) &
\end{array}
$$

$$\mathcal{M}\colon \mathit{TypEnv} \times \mathit{Expr} \times \mathit{Type} \to \mathit{Subst}$$

$$
\begin{array}{llll}
\mathcal{M}(\Gamma,\, \texttt{()},\, \rho) & = & \mathcal{U}(\rho,\, \iota) & (\mathcal{M}.1)\\
\mathcal{M}(\Gamma,\, x,\, \rho) & = & \mathcal{U}(\rho,\, \{\vec{\beta}/\vec{\alpha}\}\tau) \text{ where } \Gamma(x) = \forall\vec{\alpha}.\tau,\ \text{new } \vec{\beta} & (\mathcal{M}.2)\\
\mathcal{M}(\Gamma,\, \lambda x.e,\, \rho) & = & \text{let} \ \ S_1 = \mathcal{U}(\rho,\, \beta_1 \to \beta_2),\ \text{new } \beta_1,\, \beta_2 & (\mathcal{M}.3)\\
 & & \phantom{\text{let}} \ \ S_2 = \mathcal{M}(S_1\Gamma + x\colon S_1\beta_1,\, e,\, S_1\beta_2) & (\mathcal{M}.4)\\
 & & \text{in} \ \ S_2S_1 & \\
\mathcal{M}(\Gamma,\, e_1\ e_2,\, \rho) & = & \text{let} \ \ S_1 = \mathcal{M}(\Gamma,\, e_1,\, \beta \to \rho),\ \text{new } \beta & (\mathcal{M}.5)\\
 & & \phantom{\text{let}} \ \ S_2 = \mathcal{M}(S_1\Gamma,\, e_2,\, S_1\beta) & (\mathcal{M}.6)\\
 & & \text{in} \ \ S_2S_1 & \\
\multicolumn{3}{l}{\mathcal{M}(\Gamma, \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2, \rho) =} & \\
 & & \text{let} \ \ S_1 = \mathcal{M}(\Gamma,\, e_1,\, \beta),\ \text{new } \beta & (\mathcal{M}.7)\\
 & & \phantom{\text{let}} \ \ S_2 = \mathcal{M}(S_1\Gamma + x\colon \mathit{Clos}_{S_1\Gamma}(S_1\beta),\, e_2,\, S_1\rho) & (\mathcal{M}.8)\\
 & & \text{in} \ \ S_2S_1 & \\
\mathcal{M}(\Gamma,\, \texttt{fix}\ f\ \lambda x.e,\, \rho) & = & \mathcal{M}(\Gamma + f\colon\rho,\, \lambda x.e,\, \rho) & (\mathcal{M}.9)
\end{array}
$$

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

- 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` $\to$ `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`. ($\mathcal{W}$) $\square$

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

Our generalized algorithm is one that allows, wherever possible, the loosening of the type

$$
\begin{array}{lll}
\multicolumn{3}{c}{\mathcal{G} : \mathit{TypEnv} \times \mathit{Expr} \times \mathit{Type} \rightarrow \mathit{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) \\
\qquad\quad S_2 = \mathcal{G}(S_1\Gamma + x{:}\, S_1\beta_1, e, S_1\beta_2) & & (\mathcal{G}.4) \\
\qquad\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) \\
\qquad\quad S_2 = \mathcal{U}(S_1\theta_1, \theta_2), & (3)\ \theta_2 \geq S_1(\beta \rightarrow \rho) & (\mathcal{G}.7) \\
\qquad\quad S_3 = \mathcal{G}(S_2S_1\Gamma, e_2, \theta_3), & (4)\ \theta_3 \geq S_2S_1\beta & (\mathcal{G}.8) \\
\qquad\quad S_4 = \mathcal{U}(S_3S_2S_1\theta_1, S_3S_2S_1(\beta \rightarrow \rho)) & & (\mathcal{G}.9) \\
\qquad\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\texttt{=}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) \\
\qquad\quad S_2 = \mathcal{G}(S_1\Gamma + x{:}\, \mathit{Clos}_{S_1\Gamma}(S_1\beta), e_2, \theta), & (5)\ \theta \geq S_1\rho & (\mathcal{G}.12) \\
\qquad\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 S_1 = \mathcal{G}(\Gamma + f{:}\, \theta_1, \lambda x.e, \theta_2), & (6)\ \theta_1 \wedge \theta_2 \geq \rho & (\mathcal{G}.14) \\
\qquad\quad S_2 = \mathcal{U}(S_1\theta_1, S_1\theta_2, S_1\rho) & & (\mathcal{G}.15) \\
\quad \text{in} \quad S_2S_1 & &
\end{array}
$$

Figure 3: Generalized type inference algorithm $\mathcal{G}$. All the type variables in $\mathit{ftv}(\theta) \setminus \mathit{ftv}(\rho)$ (respectively, $(\mathit{ftv}(\theta_1) \cup \mathit{ftv}(\theta_2)) \setminus \mathit{ftv}(\rho)$) for each $\theta \geq \rho$ (respectively, $\theta_1 \wedge \theta_2 \geq \rho$) 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 $\mathit{New} \cap (\mathit{ftv}(\Gamma) \cup \mathit{ftv}(\rho)) = \emptyset$.

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

2.2 Algorithm Definition

The generalized algorithm $\mathcal{G}$ is shown in Figure 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 from applying the final substitution to the type constraint of the expression. The type constraints are just types.

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

Definition 1 ($\theta \geq \rho$) Type θ is more general (less restraining) than type ρ, written $\theta \geq \rho$, if and only if there exists a substitution G such that $G\theta = \rho$ and $supp(G) = ftv(\theta) \backslash ftv(\rho)$. We write $\theta_1 \wedge \theta_2 \geq \rho$ if and only if there exists a substitution G such that $G\theta_1 = \rho$ and $G\theta_2 = \rho$ and $supp(G) = (ftv(\theta_1) \cup ftv(\theta_2)) \setminus ftv(\rho)$.

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 expression e. It can be any type that is less restraining than the range type of ρ. We choose such a type by loosening ρ first, then picking up its range component by unification:

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

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

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

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 the 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 make up for passing less restraining type constraint. This last step is done by checking whether the loosened constraint θ can agree with the type that its original ρ becomes:

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

Consider type-checking 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 less restraining than the most informative constraint $\beta \to \rho$ with β being a new type variable:

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

After the success of this recursive call and before we continue by type-checking the argument expression, we can make up, not necessarily completely, for passing less restraining type constraint θ_1. This reparation can be varied by how much we want to expect for the type of e_1. We can check the result type against the strongest constraint $\beta \to \rho$ or we can check against nothing. This varied degree of reparation is achieved by choosing yet another less restraining type θ_2 than $S_1(\beta \to \rho)$ and by unifying it with the type that θ_1 becomes:

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

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

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

The finalizing compensation for passing the less restraining type constraints to the two recursive calls are done by checking whether the first loosened constraint θ_1 can agree with the type that the original type $\beta \to \rho$ becomes:

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

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

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

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:

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

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

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

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

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

The case for recursive function `fix` f $\lambda x.e$ is similar. We decide on what is expected for the type of $\lambda x.e$ and what is carried for the type of f. Both can be less restraining than ρ:

$$S_1 = \mathcal{G}(\Gamma + f\colon \theta_1, \lambda x.e, \theta_2), \quad (6)\ \theta_1 \wedge \theta_2 \geq \rho. \tag{$\mathcal{G}$.14}$$

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

$$S_2 = \mathcal{U}(S_1\theta_1, S_1\theta_2, S_1\rho). \tag{$\mathcal{G}$.15}$$

	(1) θ	(2) θ_1	(3) θ_2	(4) θ_3	(5) θ	(6) θ_1, θ_2
$\mathcal{W}$	β_1	β_1	β_2	β_3	β_1	β_1, β_2
SML/NJ's	β_1 or ρ	β_1	β_2	β_3	β_1	β_1, β_1
OCaml's	ρ	β_1	$S_1(\beta \to \rho)$	$S_2 S_1 \beta$	$S_1 \rho$	ρ, ρ
$\mathcal{H}$	ρ	$\beta \to \beta_2$	$S_1(\beta \to \rho)$	$S_2 S_1 \beta$	$S_1 \rho$	ρ, ρ
$\mathcal{M}$	ρ	$\beta \to \rho$	$S_1(\beta \to \rho)$	$S_2 S_1 \beta$	$S_1 \rho$	ρ, ρ

Figure 4: Five instances of algorithm $\mathcal{G}$. β_i's are new type variables introduced in the θ's.

2.3 Instances

By determining the loosened constraints θ's in $\mathcal{G}$, we obtain various type-inference algorithms, including the standard Algorithm $\mathcal{W}$, the folklore top-down algorithm $\mathcal{M}$, and the combinations of the two algorithms used in the SML/NJ[sml99] and OCaml[LRVD99] 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 loosened: for each case $\theta \geq \rho$ in $\mathcal{G}$, we choose ρ for θ.
- The OCaml's type inference algorithm 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 loosened.
- The SML/NJ's type inference algorithm is an instance of $\mathcal{G}$ where the θ at (1) (line ($\mathcal{G}$.3)) is ρ if the lambda is a recursive function, otherwise, a new type variable, the θ_1 and θ_2 at (6) (line ($\mathcal{G}$.14)) are the same new type variable, and other θ's are new type variables.
- 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 their most restraining constraints. Let's call this instance algorithm $\mathcal{H}$.

The θ's used in the five instances are summarized in Figure 4.

3 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 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 proof uses Lemmas 1 to 3 and Theorem 2.

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

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

Lemma 3 [Mil78] *Let S be a substitution, Γ be a type environment, and τ be a type. $S\,Clos_\Gamma(\tau) = Clos_{S'\Gamma}(S'\tau)$, where $S' = S\{\vec{\beta}/\vec{\alpha}\}$, $\vec{\alpha} = ftv(\tau) \setminus ftv(\Gamma)$ and $\vec{\beta}$ is new.*

Theorem 2 [Rob65] *There is an algorithm $\mathcal{U}$ which, given a pair of types, either returns a substitution S or fails; further*

- *If $S = \mathcal{U}(\tau, \tau')$ then $S\tau = S\tau'$.*
- *If S' unifies 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'.

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

- **case** `()`: $S\rho = S\iota = \iota$. So $S\Gamma \vdash \texttt{()} : S\rho$ by (CON).
- **case** x: $S\rho = S\{\vec{\beta}/\vec{\alpha}\}\tau \prec S\Gamma(x)$ by Lemma 2. So $S\Gamma \vdash x : S\rho$ by (VAR).
- **case** $\lambda x.e$: By induction hypothesis, ($\mathcal{G}$.4) implies that $S_2 S_1 \Gamma + x{:}\, S_2 S_1 \beta_1 \vdash e : S_2 S_1 \beta_2$. By (FN),
$$S_2 S_1 \Gamma \vdash \lambda x.e : S_2 S_1 (\beta_1 \to \beta_2).$$
By Lemma 1, we can apply S_3 to both sides:
$$S_3 S_2 S_1 \Gamma \vdash \lambda x.e : S_3 S_2 S_1 (\beta_1 \to \beta_2).$$
Because $S_1(\beta_1 \to \beta_2) = S_1\theta$ by ($\mathcal{G}$.3) and $S_3 S_2 S_1 \theta = S_3 S_2 S_1 \rho$ by ($\mathcal{G}$.5),
$$S_3 S_2 S_1 \Gamma \vdash \lambda x.e : S_3 S_2 S_1 \rho.$$
- **case** $e_1\ e_2$: By induction, ($\mathcal{G}$.6) implies $S_1\Gamma \vdash e_1 : S_1\theta_1$. By Lemma 1, we can apply $S_5 S_4 S_3 S_2$ to both sides:
$$S_5 S_4 S_3 S_2 S_1 \Gamma \vdash e_1 : S_5 S_4 S_3 S_2 S_1 \theta_1.$$
Because $S_4 S_3 S_2 S_1 \theta_1 = S_4 S_3 S_2 S_1 (\beta \to \rho)$ by ($\mathcal{G}$.9) and $S_5 S_4 S_3 S_2 S_1 \beta = S_5 S_4 S_3 \theta_3$ by ($\mathcal{G}$.10),
$$S_5 S_4 S_3 S_2 S_1 \Gamma \vdash e_1 : S_5 S_4 S_3 (\theta_3 \to S_2 S_1 \rho). \tag{7}$$
By induction, ($\mathcal{G}$.8) implies $S_3 S_2 S_1 \Gamma \vdash e_2 : S_3\theta_3$. By Lemma 1, we can apply $S_5 S_4$ to both sides:
$$S_5 S_4 S_3 S_2 S_1 \Gamma \vdash e_2 : S_5 S_4 S_3 \theta_3. \tag{8}$$
Hence by (APP), (7) and (8) imply
$$S_5 S_4 S_3 S_2 S_1 \Gamma \vdash e_1\ e_2 : S_5 S_4 S_3 S_2 S_1 \rho.$$
- **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 1, we can apply S_2' to both sides:
$$S_2' S_1 \Gamma \vdash e_1 : S_2' S_1 \beta. \tag{9}$$
By induction, ($\mathcal{G}$.12) implies
$$S_2 S_1 \Gamma + x{:}\, S_2\, \mathit{Clos}_{S_1\Gamma}(S_1\beta) \vdash e_2 : S_2\theta. \tag{10}$$

Note that $S_2 S_1 \Gamma = S'_2 S_1 \Gamma$ 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 3. Thus (10) is

$$S'_2 S_1 \Gamma + x\colon Clos_{S'_2 S_1\Gamma}(S'_2 S_1 \beta) \vdash e_2 : S_2\theta. \tag{11}$$

Hence by (LET), (9) and (11) imply $S'_2 S_1 \Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : S_2\theta$; that is,

$$S_2 S_1 \Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : S_2\theta.$$

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

$$S_3 S_2 S_1 \Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : S_3 S_2\theta.$$

Because $S_3 S_2 \theta = S_3 S_2 S_1 \rho$ by ($\mathcal{G}$.13),

$$S_3 S_2 S_1 \Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : S_3 S_2 S_1\rho.$$

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

$$S_2 S_1 \Gamma + f\colon S_2 S_1 \theta_1 \vdash \lambda x.e : S_2 S_1 \theta_2.$$

Because $S_2 S_1 \theta_1 = S_2 S_1 \theta_2 = S_2 S_1 \rho$ by ($\mathcal{G}$.15),

$$S_2 S_1 \Gamma + f\colon S_2 S_1 \rho \vdash \lambda x.e : S_2 S_1 \rho.$$

Hence by (FIX),

$$S_2 S_1 \Gamma \vdash \texttt{fix}\ f\ \lambda x.e : S_2 S_1 \rho. \quad \Box$$

Theorem 3 (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 $P|_{New} = (RS)|_{New}$ where New is the set of new type variables used by $\mathcal{G}(\Gamma, e, \rho)$.*

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, $\exists R.\tau = R(S\rho)$). The completeness proof uses Lemmas 4 to 8.

Lemma 4 [LY98] *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 5 [DM82] *Let Γ and Γ' be type environments such that $\Gamma \succ \Gamma'$. If $\Gamma' \vdash e : \tau$, then $\Gamma \vdash e : \tau$.*

Lemma 6 [Mil78] *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 7 *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 $\mathcal{G}(\Gamma, e, \rho)$.*

Proof. See Appendix A. □

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

Proof of Theorem 3. We prove by structural induction on e. For a rigorous treatment of new type variables, we assume that every new type variable used throughout algorithm $\mathcal{G}$ 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 [LY98].
- **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 2, the unification at ($\mathcal{G}$.3) succeeds with S_1 such that for a substitution R_1,

$$R_1 S_1 = P'. \tag{12}$$

  By the (FN) rule, the given judgment implies

$$P\Gamma + x\colon \tau_1 \vdash e : \tau_2. \tag{13}$$

  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 (13), we must prove that there exists a substitution P_1 such that $\tau_2 = P_1(S_1\beta_2)$ and $P\Gamma + x\colon \tau_1 = P_1(S_1\Gamma + x\colon S_1\beta_1)$. Such P_1 is R_1 at (12) because

$$\begin{array}{rcll} R_1(S_1\beta_2) & = & P'\beta_2 & \text{by (12)} \\ & = & \tau_2 & \text{by the definition of } P' \end{array}$$

  and

$$\begin{array}{rcll} R_1(S_1\Gamma + x\colon S_1\beta_1) & = & P'(\Gamma + x\colon \beta_1) & \text{by (12)} \\ & = & PG\Gamma + x\colon \tau_1 & \text{because the new } \beta_1, \beta_2 \notin ftv(\Gamma) \\ & = & P\Gamma + x\colon \tau_1 & \text{because } supp(G) \cap ftv(\Gamma) = \emptyset. \end{array}$$

  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,

$$(R_2 S_2)|_{New_1} = R_1|_{New_1}. \tag{14}$$

Note that

$$\begin{array}{lll} \mathit{itv}(S_1) & & \\ \quad \subseteq & \{\beta_1, \beta_2\} \cup \mathit{ftv}(\theta) & \text{by Theorem 2} \\ \quad \subseteq & \{\beta_1, \beta_2\} \cup \mathit{ftv}(\rho) \cup \mathit{supp}(G) & \text{because } \mathit{supp}(G) = \mathit{ftv}(\theta) \setminus \mathit{ftv}(\rho) \end{array}$$

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

$$\mathit{New}_1 \cap \mathit{itv}(S_1) = \emptyset. \tag{15}$$

Then

$$\begin{array}{rcll} (R_2S_2S_1)\nmid_{\mathit{New}_1} & = & (R_2S_2)\nmid_{\mathit{New}_1} S_1 & \text{by Lemma 8 and (15)} \\ & = & R_1\nmid_{\mathit{New}_1} S_1 & \text{by (14)} \\ & = & (R_1S_1)\nmid_{\mathit{New}_1} & \text{by Lemma 8 and (15)} \\ & = & P'\nmid_{\mathit{New}_1} & \text{by (12).} \end{array} \tag{16}$$

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

$$\begin{array}{rcll} R_2(S_2S_1\theta) & = & P'\theta & \text{by (16) and because } \mathit{ftv}(\theta) \cap \mathit{New}_1 = \emptyset \\ & = & PG\theta & \text{because the new } \beta_1, \beta_2 \notin \mathit{ftv}(\theta) \\ & = & P\rho & \text{by the definition of } G \\ & = & PG\rho & \text{because } \mathit{ftv}(\rho) \cap \mathit{supp}(G) = \emptyset \\ & = & P'\rho & \text{because the new } \beta_1, \beta_2 \notin \mathit{ftv}(\rho) \\ & = & R_2(S_2S_1\rho) & \text{by (16) and because } \mathit{ftv}(\rho) \cap \mathit{New}_1 = \emptyset. \end{array}$$

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

$$R_3S_3 = R_2. \tag{17}$$

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

$$\begin{array}{rcll} (R_3S_3S_2S_1)\nmid_{\mathit{New}} & = & (R_2S_2S_1)\nmid_{\mathit{New}} & \text{by (17)} \\ & = & P'\nmid_{\mathit{New}} & \text{by (16)} \\ & = & P\nmid_{\mathit{New}} & \text{because } \mathit{supp}(G) \cup \{\beta_1, \beta_2\} \subseteq \mathit{New}. \end{array}$$

- **case** $e_1\ e_2$: Let the given judgment be $P\Gamma \vdash e_1\ e_2 : P\rho$, and $\mathit{New} = \{\beta\} \cup \mathit{supp}(G_1) \cup \mathit{supp}(G_2) \cup \mathit{supp}(G_3) \cup \mathit{New}_1 \cup \mathit{New}_2$, where β is the new type variable used at ($\mathcal{G}$.6), G_1, G_2 and G_3 are respectively the substitutions for $\theta_1 \geq \beta \to \rho$ at ($\mathcal{G}$.6), $\theta_2 \geq S_1(\beta \to \rho)$ 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

  $$P\Gamma \vdash e_1 : \tau \to P\rho \tag{18}$$

  and

  $$P\Gamma \vdash e_2 : \tau. \tag{19}$$

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 (18), there exists a substitution R_1 such that

$$(R_1 S_1)|_{New_1} = (P'G_1)|_{New_1}. \tag{20}$$

Then $R_1 G_2$ unifies $S_1\theta_1$ and θ_2 at ($\mathcal{G}$.7) because, by noting that

$$\begin{array}{lll} \multicolumn{3}{l}{\mathit{ftv}(S_1\theta_1) \cap \mathit{supp}(G_2)} \\ \subseteq & (\mathit{itv}(S_1) \cup \mathit{ftv}(\theta_1)) \cap \mathit{supp}(G_2) & \text{by Lemma 6} \\ \subseteq & (\mathit{ftv}(\Gamma) \cup \mathit{New}_1 \cup \mathit{ftv}(\theta_1)) \cap \mathit{supp}(G_2) & \text{by Lemma 7} \\ = & \emptyset, & \end{array} \tag{21}$$

$$\begin{array}{lll} \multicolumn{3}{l}{R_1 G_2(S_1\theta_1)} \\ = & R_1 S_1 \theta_1 & \text{by (21)} \\ = & P'G_1\theta_1 & \text{by (20) and because } \mathit{ftv}(\theta_1) \cap \mathit{New}_1 = \emptyset \\ = & P'(\beta \to \rho) & \text{by the definition of } G_1 \\ = & P'G_1(\beta \to \rho) & \text{because } \mathit{ftv}(\beta \to \rho) \cap \mathit{supp}(G_1) = \emptyset \\ = & R_1 S_1(\beta \to \rho) & \text{by (20) and because } \mathit{ftv}(\beta \to \rho) \cap \mathit{New}_1 = \emptyset \\ = & R_1 G_2(\theta_2) & \text{by the definition of } G_2. \end{array}$$

Thus the unification at ($\mathcal{G}$.7) succeeds with S_2 such that for a substitution R_2, $R_2 S_2 = R_1 G_2$. Then

$$\begin{array}{lll} \multicolumn{3}{l}{(R_2 S_2 S_1)|_{\mathit{supp}(G_2) \cup New_1}} \\ = & (R_1 G_2 S_1)|_{\mathit{supp}(G_2) \cup New_1} & \\ = & (R_1 S_1)|_{\mathit{supp}(G_2) \cup New_1} & \text{because } \mathit{supp}(G_2) \cap \mathit{itv}(S_1) = \emptyset \\ & & \text{by Lemma 7} \\ = & (P'G_1)|_{\mathit{supp}(G_2) \cup New_1} & \text{by (20).} \end{array} \tag{22}$$

In order to apply induction to $\mathcal{G}(S_2 S_1 \Gamma, e_2, \theta_3)$ at ($\mathcal{G}$.8) and (19), we must prove that there exists a substitution P_1 such that $P_1(S_2 S_1 \Gamma) = P\Gamma$ and $P_1\theta_3 = \tau$. Such P_1 is $R_2 G_3$. First, note that, by the definition of $\mathcal{G}$,

$$\mathit{supp}(G_3) \cap \mathit{ftv}(S_2 S_1 \Gamma) = \emptyset \tag{23}$$

because

$$\begin{array}{lll} \multicolumn{3}{l}{\mathit{ftv}(S_2 S_1 \Gamma)} \\ \subseteq & \mathit{itv}(S_2) \cup \mathit{itv}(S_1) \cup \mathit{ftv}(\Gamma) & \text{by Lemma 6} \\ \subseteq & \mathit{ftv}(\theta_2) \cup \mathit{ftv}(\theta_1) \cup \mathit{New}_1 \cup \mathit{ftv}(\Gamma) & \text{by Theorem 2 and Lemma 7.} \end{array}$$

Thus

$$\begin{array}{lll} \multicolumn{3}{l}{R_2 G_3(S_2 S_1 \Gamma)} \\ = & R_2 S_2 S_1 \Gamma & \text{by (23)} \\ = & P'G_1\Gamma & \text{by (22) and because } \mathit{ftv}(\Gamma) \cap (\mathit{supp}(G_2) \cup \mathit{New}_1) = \emptyset \\ = & P\Gamma & \text{because } \mathit{ftv}(\Gamma) \cap (\{\beta\} \cup \mathit{supp}(G_1)) = \emptyset. \end{array}$$

Second,

$$\begin{array}{rcll} R_2G_3(\theta_3) & = & R_2S_2S_1\beta & \text{by the definition of } G_3 \\ & = & P'G_1\beta & \text{by (22) and because } \beta \notin \mathit{supp}(G_2) \cup \mathit{New}_1 \\ & = & P'\beta & \text{because } \beta \notin \mathit{supp}(G_1) \\ & = & \tau & \text{by the definition of } P'. \end{array}$$

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

$$(R_3S_3)\restriction_{New_2} = (R_2G_3)\restriction_{New_2}. \tag{24}$$

Moreover, note that

$$(R_3S_3)\restriction_{New_2 \cup supp(G_3)} = R_2\restriction_{New_2 \cup supp(G_3)}. \tag{25}$$

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

$$\begin{array}{lll} R_3S_3S_2S_1\theta_1 & & \\ \quad = & R_2S_2S_1\theta_1 & \text{by (25) and} \\ & & \text{because } \mathit{ftv}(\theta_1) \cap (\mathit{New}_2 \cup \mathit{supp}(G_3)) = \emptyset \\ \quad = & P'G_1\theta_1 & \text{by (22) and} \\ & & \text{because } \mathit{ftv}(\theta_1) \cap (\mathit{New}_1 \cup \mathit{supp}(G_2)) = \emptyset \\ \quad = & P'(\beta \to \rho) & \text{by the definition of } G_1 \\ \quad = & P'G_1(\beta \to \rho) & \text{because } \mathit{ftv}(\beta \to \rho) \cap \mathit{supp}(G_1) = \emptyset \\ \quad = & R_2S_2S_1(\beta \to \rho) & \text{by (22) and} \\ & & \text{because } \mathit{ftv}(\beta \to \rho) \cap (\mathit{New}_1 \cup \mathit{supp}(G_2)) = \emptyset \\ \quad = & R_3S_3S_2S_1(\beta \to \rho) & \text{by (25) and} \\ & & \text{because } \mathit{ftv}(\beta \to \rho) \cap (\mathit{New}_2 \cup \mathit{supp}(G_3)) = \emptyset. \end{array}$$

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

$$R_4S_4 = R_3. \tag{26}$$

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

$$\begin{array}{lll} R_4(S_4S_3\theta_3) & & \\ \quad = & R_3S_3\theta_3 & \text{by (26)} \\ \quad = & R_2G_3\theta_3 & \text{by (24) and because } \mathit{ftv}(\theta_3) \cap \mathit{New}_2 = \emptyset \\ \quad = & R_2S_2S_1\beta & \text{by the definition of } G_3 \\ \quad = & R_4(S_4S_3S_2S_1\beta) & \text{by (25) and (26), and} \\ & & \text{because } \beta \notin \mathit{New}_2 \cup \mathit{supp}(G_3). \end{array}$$

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

$$R_5S_5 = R_4. \tag{27}$$

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

Now we prove the rest that $(R_5S_5S_4S_3S_2S_1)\restriction_{New} = P\restriction_{New}$. Note that, by Lemma 6 and 7 and Theorem 2, $\mathit{itv}(S_2S_1) \subseteq \mathit{ftv}(\Gamma) \cup \mathit{ftv}(\theta_1) \cup \mathit{ftv}(\theta_2) \cup \mathit{New}_1$, hence by the definition of $\mathcal{G}$,

$$\mathit{itv}(S_2S_1) \cap (\mathit{New}_2 \cup \mathit{supp}(G_3)) = \emptyset. \tag{28}$$

Therefore

$$\begin{array}{ll}
(R_5S_5S_4S_3S_2S_1)\!\restriction_{New} & \\
= (R_4S_4S_3S_2S_1)\!\restriction_{New} & \text{by (27)} \\
= (R_3S_3S_2S_1)\!\restriction_{New} & \text{by (26)} \\
= ((R_3S_3)\!\restriction_{New_2 \cup supp(G_3)} S_2S_1)\!\restriction_{New} & \text{by Lemma 8 and (28)} \\
= (R_2\!\restriction_{New_2 \cup supp(G_3)} S_2S_1)\!\restriction_{New} & \text{by (25)} \\
= (R_2S_2S_1)\!\restriction_{New} & \text{by Lemma 8 and (28)} \\
= (P'G_1)\!\restriction_{New} & \text{by (22)} \\
= P\!\restriction_{New} & \text{because } (\{\beta\} \cup \mathit{supp}(G_1)) \subseteq \mathit{New}.
\end{array}$$

- **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 $\mathit{New} = \{\beta\} \cup \mathit{supp}(G) \cup \mathit{New}_1 \cup \mathit{New}_2$, 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 $\mathcal{G}(S_1\Gamma + x{:}\, \mathit{Clos}_{S_1\Gamma}(S_1\beta), e_2, \theta)$ at ($\mathcal{G}$.12).

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

  $$P\Gamma \vdash e_1 : \tau \tag{29}$$

  and

  $$P\Gamma + x{:}\, \mathit{Clos}_{P\Gamma}(\tau) \vdash e_2 : P\rho. \tag{30}$$

  Let $P' = P\!\restriction_{\{\beta\}} \cup \{\tau/\beta\}$. Then $P'\beta = \tau$ and $P'\Gamma = P\Gamma$ because $\beta \notin \mathit{ftv}(\Gamma)$. Hence by induction, $\mathcal{G}(\Gamma, e_1, \beta)$ at ($\mathcal{G}$.11) and (29) imply that there exists a substitution R_1 such that

  $$(R_1S_1)\!\restriction_{New_1} = P'\!\restriction_{New_1}. \tag{31}$$

  Note that

  $$\begin{array}{rcll}
  R_1G(S_1\Gamma) & = & R_1S_1\Gamma & \text{because } \mathit{supp}(G) \cap \mathit{ftv}(S_1\Gamma) = \emptyset \\
  & & & \text{by Lemma 6 and 7} \\
  & = & P'\Gamma & \text{by (31) and because } \mathit{ftv}(\Gamma) \cap \mathit{New}_1 = \emptyset \\
  & = & P\Gamma & \text{because the new } \beta \notin \mathit{ftv}(\Gamma),
  \end{array}$$

  and

  $$\begin{array}{cll}
  \multicolumn{3}{l}{R_1G(\mathit{Clos}_{S_1\Gamma}(S_1\beta))} \\
  \succ & \mathit{Clos}_{R_1GS_1\Gamma}(R_1GS_1\beta) & \text{by Lemma 4} \\
  = & \mathit{Clos}_{P\Gamma}(R_1S_1\beta) & \text{because } \mathit{supp}(G) \cap \mathit{ftv}(S_1\beta) = \emptyset \\
  & & \text{by Lemma 6 and 7} \\
  = & \mathit{Clos}_{P\Gamma}(P'\beta) & \text{by (31) and because } \beta \notin \mathit{New}_1 \\
  = & \mathit{Clos}_{P\Gamma}(\tau) & \text{by the definition of } P';
  \end{array}$$

  that is, $R_1G(S_1\Gamma + x{:}\, \mathit{Clos}_{S_1\Gamma}(S_1\beta)) \succ P\Gamma + x{:}\, \mathit{Clos}_{P\Gamma}(\tau)$. Then by Lemma 5 and (30),

  $$R_1G(S_1\Gamma + x{:}\, \mathit{Clos}_{S_1\Gamma}(S_1\beta)) \vdash e_2 : P\rho. \tag{32}$$

  In order to apply induction to $\mathcal{G}(S_1\Gamma + x{:}\, \mathit{Clos}_{S_1\Gamma}(S_1\beta), e_2, \theta)$ at ($\mathcal{G}$.12) and (32), we have to prove that $R_1G\theta = P\rho$:

  $$\begin{array}{rcll}
  R_1G(\theta) & = & R_1S_1\rho & \text{by the definition of } G \\
  & = & P'\rho & \text{by (31) and because } \mathit{ftv}(\rho) \cap \mathit{New}_1 = \emptyset \\
  & = & P\rho & \text{because the new } \beta \notin \mathit{ftv}(\rho).
  \end{array}$$

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

$$(R_2S_2)|_{New_2} = (R_1G)|_{New_2}. \tag{33}$$

Moreover, note that

$$(R_2S_2)|_{supp(G)\cup New_2} = R_1|_{supp(G)\cup New_2}. \tag{34}$$

Then R_2 unifies $S_2\theta$ and $S_2S_1\rho$ at ($\mathcal{G}$.13) because

$$\begin{array}{rcll} R_2(S_2\theta) & = & R_1G\theta & \text{by (33) and because } ftv(\theta)\cap New_2 = \emptyset \\ & = & R_1S_1\rho & \text{by the definition of } G \\ & = & R_2(S_2S_1\rho) & \text{by (34) and because, by Lemma 6 and 7,} \\ & & & ftv(S_1\rho)\cap(supp(G)\cup New_2) = \emptyset. \end{array}$$

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

$$R_3S_3 = R_2. \tag{35}$$

Hence, $\mathcal{G}(\Gamma, \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2, \rho)$ succeeds with $S_3S_2S_1$.

Now we prove the rest that $(R_3S_3S_2S_1)|_{New} = P|_{New}$. Note that, by Lemma 7, $itv(S_1) \subseteq ftv(\Gamma)\cup\{\beta\}\cup New_1$, hence by the definition of $\mathcal{G}$,

$$itv(S_1)\cap(supp(G)\cup New_2) = \emptyset. \tag{36}$$

Therefore

$$\begin{array}{cll} \multicolumn{2}{l}{(R_3S_3S_2S_1)|_{New}} & \\ = & (R_2S_2S_1)|_{New} & \text{by (35)} \\ = & ((R_2S_2)|_{supp(G)\cup New_2}S_1)|_{New} & \text{by Lemma 8 and (36)} \\ = & (R_1|_{supp(G)\cup New_2}S_1)|_{New} & \text{by (34)} \\ = & (R_1S_1)|_{New} & \text{by Lemma 8 and (36)} \\ = & P'|_{New} & \text{by (31)} \\ = & P|_{New} & \text{because } \beta\in New. \end{array}$$

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

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

  $$PG\Gamma + f{:}\,PG\theta_1 \vdash \lambda x.e : PG\theta_2.$$

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

  $$(R_1S_1)|_{New'} = (PG)|_{New'}. \tag{37}$$

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

  $$\begin{array}{rcll} R_1(S_1\rho) & = & PG\rho & \text{by (37) and because } ftv(\rho)\cap New' = \emptyset \\ & = & P\rho & \text{because } ftv(\rho)\cap supp(G) = \emptyset \\ & = & PG(\theta_1 \text{ or } \theta_2) & \text{by the definition of } G \\ & = & R_1(S_1\theta_1 \text{ or } S_1\theta_2) & \text{by (37) and} \\ & & & \text{because } (ftv(\theta_1)\cup ftv(\theta_2))\cap New' = \emptyset. \end{array}$$

Thus the unification at ($\mathcal{G}$.15) succeeds with S_2 such that for a substitution R_2,

$$R_2S_2 = R_1. \tag{38}$$

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

$$\begin{array}{rcll} (R_2S_2S_1)\rvert_{New} & = & (R_1S_1)\rvert_{New} & \text{by (38)} \\ & = & (PG)\rvert_{New} & \text{by (37)} \\ & = & P\rvert_{New} & \text{because } \mathit{supp}(G) \subseteq \mathit{New}. \quad \Box \end{array}$$

4 More Restraining Instance Stops Earlier

The information amount in the type constraints determines how early the algorithm detects type errors. Carrying less informative (restraining) constraints during type-checking subexpressions 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 loosening operations preserve the restraining order between the original constraints: for each pair of corresponding loosenings $\theta_i \geq \rho_i$ in A and $\theta'_i \geq \rho'_i$ in A' for the same input, if ρ_i is more restraining than ρ'_i then so is θ_i than θ'_i.

Definition 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 loosenings $\theta_i \geq \rho_i$ during $A(\Gamma, e, \rho)$ and $\theta'_i \geq \rho'_i$ during $A'(\Gamma, e, \rho)$, if $\rho_i = R\rho'_i$ for a substitution R then $\theta_i = (R\rvert_{supp(P)} \cup P)\theta'_i$ for a substitution P with $supp(P) \subseteq ftv(\theta'_i) \setminus ftv(\rho'_i)$.

Lemma 9 *$\mathcal{M} \sqsubseteq \mathcal{H} \sqsubseteq$ 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 $\theta = (R\rvert_{supp(P)} \cup P)\theta'$.

- **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 $(R\rvert_{\{\beta'_2\}} \cup \{\rho/\beta'_2\})(\beta' \to \beta'_2) = R\beta' \to \rho = \beta \to \rho$.
- **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, $(R\rvert_{\{\beta'_1\}} \cup \{\beta \to \beta_2/\beta'_1\})\beta'_1 = \beta \to \beta_2$.
- **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 $\beta'_1 \geq \rho$ or $\rho' \geq \rho'$. By the assumption, $R\rho' = \rho$. Thus $(R\rvert_{\{\beta'_1\}} \cup \{\rho/\beta'_1\})\beta'_1 = \rho$ or $(R \cup \{\})\rho' = \rho$.
  - 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, $(R\rvert_{\{\beta'_1\}} \cup \{\beta_1/\beta'_1\})\beta'_1 = \beta_1$.
  - 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, $(R\rvert_{\{\beta'_2\}} \cup \{S_1(\beta \to \rho)/\beta'_2\})\beta'_2 = S_1(\beta \to \rho)$.

  - 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 $\rho \wedge \rho \geq \rho$. For SML/NJ's, it is $\beta_1' \wedge \beta_1' \geq \rho'$. For any substitution R, $(R\!\!\restriction_{\{\beta_1'\}} \cup \{\rho/\beta_1'\})\beta_1' = \rho$.

- **case** SML/NJ's $\sqsubseteq \mathcal{W}$:
  - case (1) at ($\mathcal{G}$.3): For SML/NJ's, it is $\beta_1 \geq \rho$ or $\rho \geq \rho$. For $\mathcal{W}$, it is $\beta_1' \geq \rho'$. For any substitution R, $(R\!\!\restriction_{\{\beta_1'\}} \cup \{\beta_1/\beta_1'\})\beta_1' = \beta_1$ or $(R\!\!\restriction_{\{\beta_1'\}} \cup \{\rho/\beta_1'\})\beta_1' = \rho$.
  - case (6) at ($\mathcal{G}$.14): For SML/NJ's, it is $\beta_1 \wedge \beta_1 \geq \rho$. For $\mathcal{W}$, it is $\beta_1' \wedge \beta_2' \geq \rho'$. For any substitution R, $(R\!\!\restriction_{\{\beta_1',\beta_2'\}} \cup \{\beta_1/\beta_1', \beta_1/\beta_2'\})$ $(\beta_1'$ or $\beta_2') = \beta_1$.
  - 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, $(R\!\!\restriction_{\{\beta_i'\}} \cup \{\beta_i/\beta_i'\})\beta_i' = \beta_i$. $\square$

The time of detecting type errors can be formalized by the notion of *call string* [LY98]. 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 then A' then A stops earlier than A' if the input program is ill-typed:

Theorem 4 *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 $R\rho' = \rho$.*

Because the order of visiting sub-expressions during the execution of the two instance algorithms are the same, the above theorem implies that if A is more restraining than A' then the length (the number of tuples) $|[\![A(\Gamma_0, e_0, \rho_0)]\!]|$ 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'.

The proof of Theorem 4 uses Lemmas 10 and 11.

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

Lemma 11 *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 $(R'S')\!\!\restriction_{New} = (SR)\!\!\restriction_{New}$ where New is the set of new type variables used by $A'(\Gamma', e, \rho')$.*

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

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

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

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

By the completeness of A', $A'(\Gamma', e, \rho')$ succeeds with S' and there exists a substitution R' such that

$$(R'S')|_{New} = (SR)|_{New}. \quad \square$$

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

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

Followings are inductive cases. We first prove for the case that the string ends with a return: $(\Gamma_0, e_0, \rho_0)^d \cdots (\Gamma, e, \rho)^u$.

- **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 11, $A'(\Gamma', e, \rho')$ 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

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

  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

  $$R\Gamma' \succ \Gamma \tag{39}$$

  and $R\rho' = \rho$.

  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

  $$\theta = (R|_{supp(P)} \cup P)\theta' \tag{40}$$

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

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

  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_1R_0 unifies $\beta_1' \to \beta_2'$ and θ' at ($\mathcal{G}$.3) because

  $$\begin{array}{rcll} S_1R_0(\theta') & = & S_1(R|_{supp(P)} \cup P)\theta' & \text{because the new } \beta_1', \beta_2' \notin ftv(\theta') \\ & = & S_1\theta & \text{by (40)} \\ & = & S_1(\beta_1 \to \beta_2) & \text{by } (\mathcal{G}.3) \\ & = & S_1R_0(\beta_1' \to \beta_2') & \text{by the definition of } R_0. \end{array}$$

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

Now we prove the rest that there exists a substitution R' such that $R'(S'_1\Gamma' + x{:}\, S'_1\beta'_1) \succ (S_1\Gamma + x{:}\, S_1\beta_1)$ and $R'(S'_1\beta'_2) = S_1\beta_2$. Because $(\mathcal{G}.3)$ succeeds with S'_1, by Theorem 2, there exists a substitution R_1 such that

$$S_1 R_0 = R_1 S'_1. \tag{42}$$

Then such R' is R_1 because

$$\begin{array}{lll} R_1(S'_1\Gamma' + x{:}\, S'_1\beta'_1) & & \\ \quad = & S_1 R_0(\Gamma' + x{:}\, \beta'_1) & \text{by (42)} \\ \quad = & S_1((R|_{\overline{supp(P)}} \cup P)\Gamma' + x{:}\, R_0\beta'_1) & \text{because the new } \beta'_1, \beta'_2 \notin \mathit{ftv}(\Gamma') \\ \quad = & S_1(R\Gamma' + x{:}\, \beta_1) & \text{by (41) and the definition of } R_0 \\ \quad \succ & S_1(\Gamma + x{:}\, \beta_1) & \text{by (39) and Lemma 2} \end{array}$$

and

$$\begin{array}{rcll} R_1(S'_1\beta'_2) & = & S_1 R_0 \beta'_2 & \text{by (42)} \\ & = & S_1\beta_2 & \text{by the definition of } R_0. \end{array}$$

- **case** e **in** $e\ e_2$; 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 loosened 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

$$R\Gamma' \succ \Gamma \tag{43}$$

and $R\rho' = \rho$. 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 loosened from $\beta' \to \rho'$ at $(\mathcal{G}.8)$.

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

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

there exists a substitution P such that

$$(R_0|_{\overline{supp(P)}} \cup P)\theta'_1 = \theta_1$$

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

$$\begin{array}{rcll} (R_0|_{\overline{supp(P)}} \cup P)(\Gamma') & = & R\Gamma' & \text{because } (\{\beta\} \cup \mathit{supp}(P)) \cap \mathit{ftv}(\Gamma') = \emptyset \\ & \succ & \Gamma & \text{by (43).} \end{array}$$

- **case** e **in** $e_1\ e$; that is, $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has

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

where θ_1, θ_2, and θ_3 are respectively the loosened 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

$$R\Gamma' \succ \Gamma \tag{44}$$

and $R\rho' = \rho$.

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

  $$\begin{array}{rcll} R_0(\beta' \to \rho') & = & \beta \to R\rho' & \text{because the new } \beta' \notin \mathit{ftv}(\rho') \\ & = & \beta \to \rho, & \end{array} \tag{45}$$

  there exists a substitution P_1 such that

  $$\theta_1 = (R_0\restriction_{\mathit{supp}(P_1)} \cup P_1)\theta_1' \tag{46}$$

  and $\mathit{supp}(P_1) \subseteq \mathit{ftv}(\theta_1') \setminus \mathit{ftv}(\beta' \to \rho')$. Note that by the definition of $\mathcal{G}$,

  $$\mathit{supp}(P_1) \cap (\mathit{ftv}(\Gamma') \cup \mathit{ftv}(\beta' \to \rho')) = \emptyset \tag{47}$$

  and thus

  $$\begin{array}{rcll} (R_0\restriction_{\mathit{supp}(P_1)} \cup P_1)\Gamma' & = & R\Gamma' & \text{by (47) and because } \beta' \notin \mathit{ftv}(\Gamma') \\ & \succ & \Gamma & \text{by (44).} \end{array} \tag{48}$$

  Because $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma, e_1, \theta_1)^u$, $(R_0\restriction_{\mathit{supp}(P_1)} \cup P_1)\Gamma' \succ \Gamma$ (48), and $(R_0\restriction_{\mathit{supp}(P_1)} \cup P_1)\theta_1' = \theta_1$ (46), by Lemma 11, $A'(\Gamma', e_1, \theta_1')$ succeeds with S_1' such that for a substitution R_1,

  $$(R_1 S_1')\restriction_{\mathit{New}_1} = (S_1(R_0\restriction_{\mathit{supp}(P_1)} \cup P_1))\restriction_{\mathit{New}_1} \tag{49}$$

  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

  $$\begin{array}{rll} & R_1(S_1'(\beta' \to \rho')) & \\ = & S_1(R_0\restriction_{\mathit{supp}(P_1)} \cup P_1)(\beta' \to \rho') & \\ & \qquad \text{by (49) and because } \mathit{ftv}(\beta' \to \rho') \cap \mathit{New}_1 = \emptyset & \\ = & S_1 R_0(\beta' \to \rho') & \text{by (47)} \\ = & S_1(\beta \to \rho) & \text{by (45),} \end{array} \tag{50}$$

  there exists a substitution P_2 such that

  $$\theta_2 = (R_1\restriction_{\mathit{supp}(P_2)} \cup P_2)\theta_2' \tag{51}$$

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

  $$\begin{array}{lll} \mathit{ftv}(S_1'\theta_1') \cup \mathit{ftv}(S_1'\beta') \cup \mathit{ftv}(S_1'\Gamma') & & \\ \quad \subseteq \ \mathit{itv}(S_1') \cup \mathit{ftv}(\theta_1') \cup \{\beta'\} \cup \mathit{ftv}(\Gamma') & \text{by Lemma 6} \\ \quad \subseteq \ \mathit{New}_1 \cup \mathit{ftv}(\theta_1') \cup \{\beta'\} \cup \mathit{ftv}(\Gamma') & \text{by Lemma 7} \end{array}$$

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

$$supp(P_2) \cap (ftv(S'_1\theta'_1) \cup ftv(S'_1\beta') \cup ftv(S'_1\Gamma')) = \emptyset. \tag{52}$$

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

$$\begin{array}{lll} S_2(R_1\nmid_{supp(P_2)} \cup P_2)(S'_1\theta'_1) & & \\ \quad = & S_2R_1S'_1\theta'_1 & \text{by (52)} \\ \quad = & S_2S_1(R_0\nmid_{supp(P_1)} \cup P_1)\theta'_1 & \text{by (49) and} \\ & & \text{because } ftv(\theta'_1) \cap New_1 = \emptyset \\ \quad = & S_2S_1\theta_1 & \text{by (46)} \\ \quad = & S_2\theta_2 & \text{by } (\mathcal{G}.7) \\ \quad = & S_2(R_1\nmid_{supp(P_2)} \cup P_2)(\theta'_2) & \text{by (51).} \end{array}$$

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

Therefore $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(S'_2S'_1\Gamma', e, \theta'_3)^d$.

Now we prove the rest that there exists a substitution R' such that $R'\theta'_3 = \theta_3$ and $R'(S'_2S'_1\Gamma') \succ S_2S_1\Gamma$. Because ($\mathcal{G}$.7) succeeds with S'_2, by Theorem 2, there exists a substitution R_2 such that

$$R_2S'_2 = S_2(R_1\nmid_{supp(P_2)} \cup P_2). \tag{53}$$

Because $A \sqsubseteq A'$ and

$$\begin{array}{rcll} R_2(S'_2S'_1\beta') & = & S_2(R_1\nmid_{supp(P_2)} \cup P_2)S'_1\beta' & \text{by (53)} \\ & = & S_2R_1S'_1\beta' & \text{by (52)} \\ & = & S_2S_1\beta & \text{by (50),} \end{array}$$

there exists a substitution P_3 such that

$$\theta_3 = (R_2\nmid_{supp(P_3)} \cup P_3)\theta'_3$$

and $supp(P_3) \subseteq ftv(\theta'_3) \setminus ftv(S'_2S'_1\beta')$. Note again that, by Lemma 6 and 7 and Theorem 2,

$$\begin{array}{rcl} ftv(S'_2S'_1\Gamma') & \subseteq & ftv(\theta_1) \cup ftv(\theta_2) \cup New_1 \cup ftv(\Gamma') \\ & \subseteq & supp(P_1) \cup ftv(\beta \to \rho) \cup supp(P_2) \cup New_1 \cup ftv(\Gamma') \end{array}$$

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

$$supp(P_3) \cap ftv(S'_2S'_1\Gamma') = \emptyset. \tag{54}$$

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

$$\begin{array}{lll} (R_2\nmid_{supp(P_3)} \cup P_3)(S'_2S'_1\Gamma') & & \\ \quad = & R_2S'_2S'_1\Gamma' & \text{by (54)} \\ \quad = & S_2(R_1\nmid_{supp(P_2)} \cup P_2)S'_1\Gamma' & \text{by (53)} \\ \quad = & S_2R_1S'_1\Gamma' & \text{by (52)} \\ \quad = & S_2S_1(R_0\nmid_{supp(P_1)} \cup P_1)\Gamma' & \text{by (49) and because } ftv(\Gamma') \cap New_1 = \emptyset \\ \quad \succ & S_2S_1\Gamma & \text{by (48) and Lemma 2.} \end{array}$$

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

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

where β is the new type variable introduced at ($\mathcal{G}$.11). By induction, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma', \texttt{let}\ x\ \texttt{=}\ e\ \texttt{in}\ e_2, \rho')^d$ 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 $R_0 = R\!\restriction_{\{\beta'\}} \cup \{\beta/\beta'\}$. Then $R_0\Gamma' = R\Gamma' \succ \Gamma$ and $R_0\beta' = \beta$.

- **case** e **in** ($\texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e$); that is, $[\![A(\Gamma_0, e_0, \rho_0)]\!]$ has

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

where β is the new type variable introduced at ($\mathcal{G}$.11), θ is the loosened 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 $(\Gamma', \texttt{let}\ x\ \texttt{=}\ e\ \texttt{in}\ e_2, \rho')^d$ and there exists a substitution R such that

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

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

$$\begin{array}{rcll} R_0\Gamma' & = & R\Gamma' & \text{because the new } \beta' \notin \mathit{ftv}(\Gamma') \\ & \succ & \Gamma & \text{by (55)} \end{array} \tag{56}$$

and $R_0\beta' = \beta$. Thus by Lemma 11, $A'(\Gamma', e_1, \beta')$ at ($\mathcal{G}$.11) succeeds with S_1', hence $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(S_1'\Gamma' + x\colon \mathit{Clos}_{S_1'\Gamma'}(S_1'\beta'), e_2, \theta')^d$.

Now we prove the rest that there exists a substitution R' such that $R'\theta' = \theta$ and $R'(S_1'\Gamma' + x\colon \mathit{Clos}_{S_1'\Gamma'}(S_1'\beta')) \succ S_1\Gamma + x\colon \mathit{Clos}_{S_1\Gamma}(S_1\beta)$. Because ($\mathcal{G}$.11) succeeds with S_1', by Lemma 11, there is a substitution R_1 such that

$$(R_1S_1')\!\restriction_{\mathit{New}_1} = (S_1R_0)\!\restriction_{\mathit{New}_1} \tag{57}$$

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

Because $A \sqsubseteq A'$ and

$$\begin{array}{rcll} R_1(S_1'\rho') & = & S_1R_0\rho' & \text{by (57) and because } \mathit{ftv}(\rho') \cap \mathit{New}_1 = \emptyset \\ & = & S_1R\rho' & \text{because the new } \beta' \notin \mathit{ftv}(\rho') \\ & = & S_1\rho, & \end{array}$$

there exists a substitution P such that

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

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

$$\begin{array}{rcll} \mathit{ftv}(S_1'\Gamma') \cup \mathit{ftv}(S_1'\beta') & \subseteq & \mathit{itv}(S_1') \cup \mathit{ftv}(\Gamma') \cup \{\beta'\} & \text{by Lemma 6} \\ & \subseteq & \mathit{New}_1 \cup \mathit{ftv}(\Gamma') \cup \{\beta'\} & \text{by Lemma 7} \end{array}$$

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

$$\mathit{supp}(P) \cap (\mathit{ftv}(S_1'\Gamma') \cup \mathit{ftv}(S_1'\beta')) = \emptyset. \tag{58}$$

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

$$\begin{array}{lll} (R_1\!\restriction_{supp(P)} \cup P)(S'_1\Gamma') & & \\ = & R_1(S'_1\Gamma') & \text{by (58)} \\ = & S_1R_0\Gamma' & \text{by (57) and because } \mathit{New}_1 \cap \mathit{ftv}(\Gamma') = \emptyset \\ \succ & S_1\Gamma & \text{by (56) and Lemma 2} \end{array} \tag{59}$$

and

$$\begin{array}{lll} (R_1\!\restriction_{supp(P)} \cup P)(\mathit{Clos}_{S'_1\Gamma'}(S'_1\beta')) & & \\ = & R_1\,\mathit{Clos}_{S'_1\Gamma'}(S'_1\beta') & \text{by (58)} \\ \succ & \mathit{Clos}_{R_1S'_1\Gamma'}(R_1S'_1\beta') & \text{by Lemma 4} \\ \succ & \mathit{Clos}_{S_1\Gamma}(R_1S'_1\beta') & \text{by (59) and Lemma 10} \\ = & \mathit{Clos}_{S_1\Gamma}(S_1R_0\beta') & \text{by (57) and because } \beta' \notin \mathit{New}_1 \\ = & \mathit{Clos}_{S_1\Gamma}(S_1\beta) & \text{by the definition of } R_0. \end{array}$$

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

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

where θ_1 and θ_2 are the loosened types at $(\mathcal{G}.14)$. 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

$$R\Gamma' \succ \Gamma \tag{60}$$

and $R\rho' = \rho$. By the definition of $\mathcal{G}$, $[\![A'(\Gamma_0, e_0, \rho_0)]\!]$ has $(\Gamma' + f\!:\theta'_1, \lambda x.e, \theta'_2)^d$.

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

$$\theta_1 = (R\!\restriction_{supp(P)} \cup P)\theta'_1 \tag{61}$$

and

$$\theta_2 = (R\!\restriction_{supp(P)} \cup P)\theta'_2.$$

Therefore

$$\begin{array}{lll} (R\!\restriction_{supp(P)} \cup P)(\Gamma' + f\!:\theta'_1) & & \\ = & R\Gamma' + f\!:\theta_1 & \text{by (61) and because } \mathit{ftv}(\Gamma') \cap \mathit{supp}(P) = \emptyset \\ \succ & \Gamma + f\!:\theta_1 & \text{by (60).} \quad \Box \end{array}$$

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

Corollary 1 *Let Γ be a type environment, e be an expression and ρ be a type.*

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

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

5 Conclusion

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

We presented a generalized let-polymorphic type inference algorithm, from which, by changing its degree of context-sensitivity, various hybrid algorithms can be instantiated. We proved that any of $\mathcal{G}$'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. Thus, every instance algorithm's soundness, completeness, and relative earliness in detecting type errors follow automatically. 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[sml99] and OCaml[LRVD99].

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

In general settings [Hen93, CU96, AW93, Tha94, Rém92] where we view type inference algorithms consist 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. We delay the constraint-solving by passing loosened constraints to recursive calls, and then solve the delayed constraints by applying posterior unifications.

Appendix

A Proof of Lemma 7

We prove by structural induction on e.

- **case** `()`: By Theorem 2, $\mathit{itv}(\mathcal{U}(\rho,\iota)) \subseteq \mathit{ftv}(\rho) \cup \mathit{ftv}(\iota) = \mathit{ftv}(\rho)$.
- **case** x:

$$\begin{array}{rcll} \mathit{itv}(\mathcal{U}(\rho, \{\vec{\beta}/\vec{\alpha}\}\tau)) & \subseteq & \mathit{ftv}(\rho) \cup \mathit{ftv}(\{\vec{\beta}/\vec{\alpha}\}\tau) & \text{by Theorem 2} \\ & \subseteq & \mathit{ftv}(\rho) \cup (\mathit{ftv}(\tau) \setminus \vec{\alpha}) \cup \vec{\beta} & \\ & = & \mathit{ftv}(\rho) \cup \mathit{ftv}(\forall\vec{\alpha}.\tau) \cup \vec{\beta} & \\ & = & \mathit{ftv}(\rho) \cup \mathit{ftv}(\Gamma(x)) \cup \vec{\beta} & \text{because } \Gamma(x) = \forall\vec{\alpha}.\tau \\ & \subseteq & \mathit{ftv}(\rho) \cup \mathit{ftv}(\Gamma) \cup \vec{\beta}. & \end{array}$$

  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.

$$\begin{array}{lll} \mathit{itv}(S_1) & \subseteq \mathit{ftv}(\theta) \cup \mathit{ftv}(\beta_1 \to \beta_2) & \text{by Theorem 2} \\ & \subseteq \mathit{ftv}(\rho) \cup \mathit{supp}(G) \cup \{\beta_1, \beta_2\} & \text{because } \mathit{supp}(G) = \mathit{ftv}(\theta) \setminus \mathit{ftv}(\rho), \end{array}$$

$$\begin{array}{llll} \mathit{itv}(S_2) & \subseteq & \mathit{ftv}(S_1\Gamma) \cup \mathit{ftv}(S_1\beta_1) \cup \mathit{ftv}(S_1\beta_2) \cup \mathit{New}_1 & \text{by induction} \\ & \subseteq & \mathit{itv}(S_1) \cup \mathit{ftv}(\Gamma) \cup \{\beta_1, \beta_2\} \cup \mathit{New}_1 & \text{by Lemma 6} \end{array}$$

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

$$\begin{array}{llll} \mathit{itv}(S_3) & \subseteq & \mathit{ftv}(S_2S_1\theta) \cup \mathit{ftv}(S_2S_1\rho) & \text{by Theorem 2} \\ & \subseteq & \mathit{itv}(S_2) \cup \mathit{itv}(S_1) \cup \mathit{ftv}(\theta) \cup \mathit{ftv}(\rho) & \text{by Lemma 6} \\ & \subseteq & \mathit{itv}(S_2) \cup \mathit{itv}(S_1) \cup \mathit{supp}(G) \cup \mathit{ftv}(\rho). & \end{array}$$

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. □

References

[AW93] Alexander Aiken and Edward L. Wimmers. Type inclusion constraints and type inference. In *Proceedings of Functional Programming Languages and Computer Architecture*, pages 31–41, 1993.

[CU96] Kenta Cho and Kazunori Ueda. Diagnosing non-well-moded concurrent logic programs. In *Joint International Conference on Logic Programming*, pages 215–229. MIT Press, 1996.

[DM82] Luis Damas and Robin Milner. Principal type-scheme for functional programs. In *Proceedings of the 9th Annual ACM Symposium on Principles of Programming Languages*, pages 207–212, New York, 1982. ACM Press.

[Hen93] Fritz Henglein. Type inference with polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):253–289, April 1993.

[LRVD99] Xavier Leroy, Didier Rémy, Jérôme Vouillon, and Damien Doligez. The objective caml system release 2.04. Institut National de Recherche en Informatique et en Automatique, November 1999. `http://caml.inria.fr`.

[LY98] Oukseh Lee and Kwangkeun Yi. Proofs about a folklore let-polymorphic type inference algorithm. *ACM Transactions on Programming Languages and Systems*, 20(4):707–723, July 1998.

[McA98] Bruce J. McAdam. On the unification of substitutions in type inference. In Kevin Hammond, Anthony J. T. Davie, and Chris Clack, editors, *Proceedings of The International Workshop on Implementation of Fuctional Languages*, volume 1595 of *Lecture Notes in Computer Science*, pages 139–154. Springer-Verlag, September 1998.

[Mil78] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[Rém92] Didier Rémy. 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.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[sml99] The Standard ML of New Jersey, release 110.0.6. Bell Labs, Lucent Technologies, November 1999. `http://cm.bell-labs.com/cm/cs/what/smlnj`.

[Tha94] Satish R. Thatte. Type inference with partial types. *Theoretical Computer Science*, 124(1):127–148, February 1994.