Let-다형성 타입 유추 알고리즘 M의 병목을 해소하기 위한 혼성 알고리즘 H

(A Hybrid Algorithm that Eliminates the Bottleneck of the Let-Polymorphic Type-Inference Algorithm M)

주 상 현 † 이 욱 세 †† 이 광 근 †††
(Sanghyun Joo) (Ouksehe Lee) (Kwangkeun Yi)

요 약 Hindley/Milner let-다형성 타입체계(let-polymorphic type system)에는 두 가지 타입 유추(type-inference) 알고리즘이 있다: 하나는 표준으로 알려진 W 알고리즘으로 프로그램의 문맥에 상관없이 상향식으로 유추하는 알고리즘이고, 다른 하나는 프로그램의 문맥에 따라 하향식으로 유추하는 M 알고리즘이다. 본 연구에서는 함수 적용(application)이 중첩되는 경우, M 알고리즘에 병목현상이 발생함을 보이고, 이러한 병목현상이 발생하지 않는 혼성 알고리즘 H를 제시한다. H 알고리즘은 M 알고리즘을 함수 적용 부분만 W 알고리즘으로 변형한 알고리즘으로, W보다는 일찍 M보다는 늦게 오류를 감지함을 보인다.

Abstract *The Hindley/Milner let-polymorphic type system has two different type inference algorithms: one is em de facto standard algorithm W that is context-insensitive, and the other is context-sensitive algorithm M. We present the bottleneck of the M algorithm in the case of repeated applications, and propose a hybrid algorithm, named H, for its remedy. The H algorithm is made out of M adopting W on the application part. We show that H stops still earlier than W but later than M when the input program has type errors.*

1. 서 론

Hindley/Milner let-다형성 타입체계(let-polym orphictype system)에는 타입 유추(type inference) 알고리즘으로 W 알고리즘[1, 2]과 M 알고리즘[3]이 있다. W 알고리즘은 Hindley/Milner 타입체계를 처음으로 충실히 구현한(sound and complete) 타입 유추 알고리즘으로 표준으로 사용되어 왔고, M은 Caml 6.0[4]에서 이론적 검증 없이 사용되다가, W의 오류 보고를 개선하기 위해서 타입 유추 알고리즘을 고안하던 Lee와 Yi에 의해 그 안전성(soundness)과 완전성(completeness)이 증명되었다[3]. W 알고리즘과 M 알고리즘은 타입 유추 방식에서 차이를 나타낸다. W는 상향식(bottom-up) 알고리즘으로 프로그램의 문맥에 영향을 받지 않고 (context-insensitive) 타입을 유추하고, M은 이와는 반대로 하향식(top-down)으로 프로그램의 문맥에 따라 (context-sensitive) 타입을 유추한다.

M 알고리즘은 오류를 일찍 감지하는 장점이 있는 반면, 함수 적용(application)이 중첩되는 경우 병목현상이 생긴다. 항등 함수(identity function)가 중첩 적용된 프로그램에서:

$$(\lambda x.x)\ (\lambda x.x)\ (\lambda x.x)\ (\lambda x.x) \cdots (\lambda x.x)$$

M은 문맥에 따라서 타입을 유추하므로, 전체 항등 함수의 개수가 n이라고 하면 첫 번째 항등 함수의 인자의 개수는 n-1 개이고 두 번째는 n-2 개라는 정보를 이용하여 각 항등 함수의 타입을 자신의 인자의

· 본 연구는 과학기술부 창의적 연구진흥사업 추진으로 얻어진 결과임을 밝힙니다.

† 비 회 원 : SK텔레콤 중앙연구원 연구원
hartjoo@sktelecom.com
†† 비 회 원 : 한국과학기술원 전자전산학과
cookcu@ropas.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
kwang@cs.kaist.ac.kr

논문접수 : 2000년 1월 4일
심사완료 : 2000년 10월 5일

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

(a) W의 오류 보고

```
#let rec fac = fun n -> if n = 0 then 1 else n * fac (n=1);;
#                                                  ^
# Expression of type 'a -> 'a -> bool
# cannot be used with type 'a -> 'a -> int
```

(b) M의 오류 보고

```
#let rec fac = fun n -> if n = 0 then 1 else n * fac (n=1);;
#                                                  ^^^
# Expression of type bool
# cannot be used with type int
```

(c) H의 오류 보고

그림 1 서로 다른 오류 보고

개수만큼의 큰 타입으로 강요한다. 결국 M 알고리즘은 각 항등 함수에서 인자의 개수에 비례하는 작업을 하고, 이를 합하면 $(n-1)+(n-2)+\cdots+2+1$ 에 비례하는, 즉 프로그램 길이의 제곱에 비례하는 작업을 한다 ($O(n^2)$). 이와는 달리 W는 각 항등 함수의 타입을 개별적으로 유추하기 때문에 프로그램 길이에 비례하는 작업만이 필요하다($O(n)$).

본 연구에서는 이와 같은 M의 병목을 해소하는 타입 유추 알고리즘 H를 제안한다. H 알고리즘은 W와 M을 선택적으로 병행하는(hybrid) 알고리즘이다. M의 병목현상이 함수 적용(application)부분에 있으므로 H 알고리즘은 함수 적용시 W와 유사하게 유추한다. 즉, 함수 적용 "$e_1\ e_2$"의 경우, H는 e_1의 타입을 단지 함수라고만 강요함으로써 병목을 해소하게 된다. e_1의 타입을 온전히 유추한 뒤 $e_1\ e_2$에 강요된 타입에 비추어서 검사를 하게 된다. 이렇게 늦게 검사를 하게 되므로 M 보다는 늦게 오류를 감지하게 된다. 그렇지만, H는 M으로부터 만들어졌기 때문에 W 보다는 일찍 오류를 감지한다.

타입 유추 알고리즘 H는 W와 M과는 다르게 오류를 보고한다. 그림 1에서 타입 오류가 있는 프로그램에 대해 세 알고리즘이 다르게 오류를 보고한다. fac은 재귀 호출 함수(recursive function)로, 함수정의 내부에서 "fac(n-1)"라고 써야하는 것을 "fac(n=1)"로 틀리게 작성하였다. W로 타입을 유추하면, 함수 전체의 유추된 타입 int→int와 함수정의 내부에서 사용된 fac의 유추된 타입 bool→int를 동일화(unification) 할 수 없어 전체 프로그램을 오류로 보고한다. 반면에 M에서는 강요된 타입을 "="에서 더 이상 동일화 할 수 없다고 보고한다. H에서는 이와는 달리 n=1에 강요된 타입 int (fac은 int 타입을 인자로 받는다.)와 n=1에서 유추된 타입 bool을 동일화 할 수 없다고 보고한다.

2. W 알고리즘과 M 알고리즘의 정의

언어와 타입

Expr	e	::=	$()$	상수
		|	x	변수
		|	$\lambda x.e$	함수정의
		|	$e\ e$	함수 적용
		|	$\texttt{let}\ x = e\ \texttt{in}\ e$	let-묶음
		|	$\texttt{fix}\ f\ \lambda x.e$	재귀 함수정의
Type	τ	::=	ι	상수 타입
		|	α	타입 변수
		|	$\tau \to \tau$	함수 타입
TypeScheme	σ	::=	$\tau \mid \forall \vec{\alpha}.\sigma$	
TypeEnv	Γ	$\in$	$Var \xrightarrow{fin} TypeScheme$	타입 환경

타입 유추 규칙

$$\text{(CON)}\quad \Gamma \vdash () : \iota$$

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

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

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

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

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

그림 2 언어와 타입 유추 규칙

그림 2는 언어와 Hindley/Milner 다형성 타입체계이고 그림 3은 W와 M이다. 그림에서 사용된 수식들은 다음과 같다.

$\vec{\alpha}$는 타입 변수의 집합 $\{\alpha_1, \cdots, \alpha_n\}$을 나타내고, 타입스킴(type scheme) $\forall \vec{\alpha}.\tau$는 $\forall \alpha_1 \cdots \alpha_n.\tau$를 나타낸다. 타입스킴이 같다는 것은 구속 변수들(bound variables)의 이름 변화에 영향 받지 않고 같다는 것을 뜻한다. 즉, $\forall \alpha.\alpha \to \iota$와 $\forall \beta.\beta \to \iota$는 같은 타입스킴이다.

자유 타입 변수(free type variable)는 타입스킴의 한정기호(quantifier)에 의해 묶이지 않는(unbounded) 타입 변수들을 의미한다. 자유 타입 변수를 구하는 함수 ftv는 다음과 같이 정의할 수 있다.

$$
\begin{array}{lcll}
ftv(\iota) & = & \emptyset & \text{상수 타입} \\
ftv(\alpha) & = & \{\alpha\} & \text{타입 변수} \\
ftv(\tau_1 \to \tau_2) & = & ftv(\tau_1) \cup ftv(\tau_2) & \text{함수 타입} \\
ftv(\forall \vec{\alpha}.\tau) & = & ftv(\tau) \setminus \vec{\alpha} & \text{타입스킴} \\
ftv(\Gamma) & = & \bigcup_{x \in dom(\Gamma)} ftv(\Gamma(x)) & \text{타입 환경}
\end{array}
$$

이 때, $dom(\Gamma) \subseteq Var$는 타입 환경(type environment) Γ의 정의역(domain)을 의미한다.

타입 치환 함수(substitution) $S\ (\in TypeVar \to Type)$는 타입 변수를 다른 타입으로 치환해 주는 함수로 정의역이 확장되어 사용된다.

$$
\begin{array}{lcl}
S\iota & = & \iota \\
S\alpha & = & \begin{cases} \tau, & \tau/\alpha \in S\text{일 때} \\ \alpha, & \text{그렇지 않을 때} \end{cases} \\
S(\tau_1 \to \tau_2) & = & S\tau_1 \to S\tau_2 \\
S(\forall \vec{\alpha}.\tau) & = & \forall \vec{\beta}.S\{\vec{\beta}/\vec{\alpha}\}\tau \quad \text{단, } \vec{\beta} \cap (itv(S) \cup ftv(\forall \vec{\alpha}.\tau)) = \emptyset \\
S\Gamma & = & \{x \mapsto S\sigma \mid x \mapsto \sigma \in \Gamma\}
\end{array}
$$

이 때, 집합 $supp(S)$는 S로 인해 치환되는 타입 변수의 집합 $\{\alpha \mid S\alpha \neq \alpha\}$이고, 집합 $itv(S)$는 S에 연루된 타입 변수(involved type variable)의 집합 $\{\alpha \mid \beta \in supp(S), \alpha \in ftv(S\beta)\} \cup supp(S)$이다. 그리고, $\{\vec{\tau}/\vec{\alpha}\}$는 타입 치환 함수 $\{\tau_i/\alpha_i \mid 1 \le i \le n\}$를, $R\vec{\alpha}$는 $\{R\alpha_1, \cdots, R\alpha_n\}$를 줄여 쓴 것이다. 타입 치환 함수 S에 R을 중첩한 타입 치환 함수 RS는 $\{R(S\alpha)/\alpha \mid \alpha \in supp(S)\} \cup \{R\alpha/\alpha \mid \alpha \in supp(R) \setminus supp(S)\}$를 의미한다. 타입 치환 함수 S와 R이 같다는 것은 모든 $\alpha \in supp(S) \cup supp(R)$에 대해서 $S\alpha = R\alpha$를 의미한다.

$P\dagger_V$는 P와 같되 V에 속하는 타입 변수를 지원하지 않는 타입 치환 함수이다:

$$P\dagger_V \stackrel{def}{=} \{\tau/\alpha \in P \mid \alpha \notin V\}.$$

"$\Gamma + x:\sigma$"는 타입 환경(type environment) Γ에 x의 항만을 σ로 대치시킨 타입 환경 $\{y \mapsto \sigma \mid x \neq y, (y \mapsto \sigma') \in \Gamma\} \cup \{x \mapsto \sigma\}$를 말한다.

구체화(instantiation) $\sigma > \tau$는 타입 τ가 타입스킴 σ로부터 얻어지는 하나의 구체적인 타입임을 말한다:

$$\forall \vec{\alpha}.\tau' \succ \tau \stackrel{def}{=} \exists S.\ S\tau' = \tau \wedge supp(S) \subseteq \vec{\alpha}.$$

예를 들면, $\forall \alpha.\alpha \to \alpha > \text{int} \to \text{int}$이다. 일반화(generalization) $Clos_\Gamma(\tau)$는 Damas와 Milner의 Gen (Γ, τ)[2]와 같이, 타입에 한정자를 붙여 일반적인 타입을 만드는 것으로 $\vec{\alpha} = ftv(\tau) \setminus ftv(\Gamma)$일 때 $\forall \vec{\alpha}.\tau$로 정의된다.

동일화 알고리즘 $U\ (\in Type \times Type \to Subst)$는 입력된 두 타입을 동일화하는 함수로 다음 성질을 만족한다.

정리 1[5]. *두 타입을 입력으로 받아 다음을 만족하는 타입 치환 함수를 주는 U 알고리즘은 존재한다.*

• $U(\tau, \tau')$가 성공하여 S를 결과로 주면 S는 τ와 τ'을 같은 타입으로 만들어 준다. 즉, $S\tau = S\tau'$이다.

• τ와 τ'을 같은 타입으로 만드는 타입 치환 함수 S'이 존재하면, $U(\tau, \tau')$은 성공하고, 결과 S에 대해 $S' = RS$인 타입 치환 함수 R이 존재한다. 이러한 S를 가장 일반적인 동일화 함수(most general unifier)라 한다.

게다가 $U(\tau, \tau')$는 τ와 τ'의 자유 타입 변수에만 연루되어 있다.

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

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

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

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

그림 3 W 알고리즘과 M 알고리즘. 매번 새로 만들어지는 타입 변수들은 서로 다르다. 그리고, 각각의 $W(\Gamma, e)$를 재귀 호출하는 경우, 새로 만들어지는 타입 변수들의 집합 V는 $V \cap ftv(\Gamma) = \emptyset$을 만족한다. 마찬가지로 각각의 $M(\Gamma, e, \rho)$을 재귀 호출하는 경우도 $V \cap (ftv(\Gamma) \cup ftv(\rho)) = \emptyset$을 만족한다.

3. M의 병목

M 알고리즘은 함수 적용이 중첩되는 경우 병목현상이 발생한다. 항등 함수를 중첩 적용한 다음의 예를 살

펴보자:

$$(\lambda x.x)\ (\lambda x.x)\ (\lambda x.x)\ (\lambda x.x)\cdots(\lambda x.x)$$

항등 함수의 개수가 n이라 하면, M은 첫 번째 항등 함수를 유추할 때 n-1 개의 인자를 갖는다고 강요한다. 두 번째 항등 함수를 유추할 때 n-2 개, 세 번째 항등 함수는 n-3 개의 인자를 갖는다고 강요한다. 이와 같이 모든 함수들에 대해서 인자의 개수를 알고 있어 인자의 개수 크기의 타입으로 강요를 하게 되어, 결국 전체 타입 유추에 드는 비용은 $O(n^2)$이 된다. 반면에 W는 각 항등 함수를 독립적으로 유추하므로 각각의 함수의 타입 유추 비용은 일정하게 된다. 결국 $O(n)$의 비용만이 든다.

병목현상의 원인은 함수 적용(application) 부분에 있다:

$$\mathcal{M}(\Gamma, e_1, \beta \to \rho),\ \text{new}\beta \qquad \text{(M.5)}$$

현재 강요된 타입 ρ를 함수 부분 e_1에 증가된 형태인 $\beta \to \rho$로 강요한다. 함수 적용이 계속 되게 되면 강요 타입은 계속 증가되어서 실행비용이 증가한다.

항등 함수를 중첩 적용한 프로그램의 자세한 타입 유추 과정을 살펴보자(그림 4). 첫 번째 열은 입력 프로그램이고 밑줄 친 부분은 각 재귀 호출에서 현재 처리되

e	ρ	$\mathcal{U}$	
$\underline{id}\ id\ \underline{id}\ id$	δ		$(a.1)$
$\underline{id}\ \underline{id}\ \underline{id}\ id$	$\gamma \to \delta$		$(a.2)$
$\underline{id}\ \underline{id}\ id\ id$	$\beta \to \gamma \to \delta$		$(a.3)$
$\underline{id}\ id\ id\ id$	$\alpha \to \beta \to \gamma \to \delta$		$(a.4)$
$(\lambda x.\underline{x})\ id\ id\ id$	$\beta \to \gamma \to \delta$	$\mathcal{U}(\beta \to \gamma \to \delta, \alpha)$	$(a.5)$
$id\ \underline{id}\ id\ id$	$\beta \to \gamma \to \delta$		$(a.6)$
$id\ (\lambda x.\underline{x})\ id\ id$	$\gamma \to \delta$	$\mathcal{U}(\gamma \to \delta, \beta)$	$(a.7)$
$id\ id\ \underline{id}\ id$	$\gamma \to \delta$		$(a.8)$
$id\ id\ (\lambda x.\underline{x})\ id$	δ	$\mathcal{U}(\delta, \gamma)$	$(a.9)$

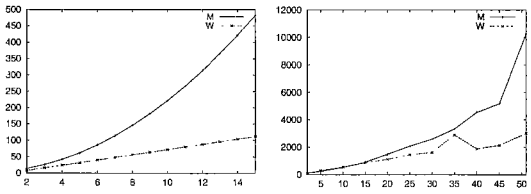
⋮

(a) $\mathcal{M}$ 에서 유추과정

e	τ	$\mathcal{U}$	
⋮			
$\underline{id}\ id\ id\ id$	$\alpha \to \alpha$		$(b.1)$
$id\ \underline{id}\ id\ id$	$\beta \to \beta$		$(b.2)$
$\underline{id}\ \underline{id}\ id\ id$	$\beta \to \beta$	$\mathcal{U}(\alpha \to \alpha, (\beta \to \beta) \to \gamma)$	$(b.3)$
$id\ id\ \underline{id}\ id$	$\delta \to \delta$		$(b.4)$
$\underline{id}\ \underline{id}\ \underline{id}\ id$	$\delta \to \delta$	$\mathcal{U}(\beta \to \beta, (\delta \to \delta) \to \theta)$	$(b.5)$
⋮			

(b) $\mathcal{W}$ 에서 유추과정

그림 4 항등 함수를 4개 중첩 적용했을 때의 타입 유추 과정



(a) 타입크기 (b) 실행시간 (μ-sec)

그림 5 항등 함수를 중첩 적용했을 때, M이 W보다 빨리 증가한다. (X: 항등 함수의 개수, Y: 측정치)

고 있는 부분이다. 두 번째 열은, M에서는 현재 처리되고 있는 부분에 강요된 타입이고, W에서는 결과 타입이다. 세 번째 열은 각 타입 유추 과정에서 불려지는 동일화 알고리즘의 인자를 보여준다. 우선, 4.(a)의 M의 유추 과정을 살펴보자. (a.1-4)에서, M은 강요 타입을 δ에서 $\alpha \to \beta \to \gamma \to \delta$로 증가시킨다. (a.5) 에서 함수 내용(function body)에 강요된 타입은 $\beta \to \gamma \to \delta$가 되고 이렇게 커진 강요 타입은 (M.2) 에 의해 α와 동일화된다. 왜냐하면, 항등 함수의 인자 타입과 결과 타입은 같아야 하기 때문이다. 인자 타입 α가 결과 타입 $\beta \to \gamma \to \delta$로 동일화되었으므로, 두 번째 항등 함수에는 $\beta \to \gamma \to \delta$가 강요된다(a.6). 나머지 과정은 위의 과정과 비슷하게 동작한다. 이와 같이 동일화가 다루는 타입의 크기는 항등 함수의 개수로부터 점차로 줄어들게 된다. 이제 W의 과정(그림 4.(b))을 살펴보자. (b.1), (b.2), (b.4) 에서 보면, W는 각 항등 함수의 타입을 $\alpha \to \alpha$와 같은 모양으로 유추한다. 이는 W가 프로그램의 문맥에 상관없이 타입을 유추하기 때문이다. 이와 같이 일정한 모양의 타입으로 (b.3), (b.5) 에서 동일화한다.

그림 5의 실험결과로부터 M의 실행비용이 프로그램의 크기가 n이라 했을 때 $O(n^2)$으로 증가하는 것을 확인할 수 있다. 반면에 W는 $O(n)$으로 증가한다. 여기서 측정한 실행비용은 동일화가 처리하는 타입의 총 크기와 실제 타입 유추 알고리즘의 실행시간이다.

4. H 알고리즘

M의 병목을 해소하기 위해 함수 적용 부분을 변경시켜 H 알고리즘을 고안하였다. 그림 6은 H 알고리즘의 정의이다.

H는 M의 함수 적용 부분에서 발생하는 병목을 해결한다. (H.5)를 보면,

$$S_1 = \mathcal{H}(\Gamma, e_1, \beta_1 \to \beta_2),\ \text{new}\ \beta_1, \beta_2 \qquad \text{(H.5)}$$

H는 함수 부분 e_1의 타입을 단지 함수라고만 즉,

$\mathcal{H}$: $TypEnv \times Expr \times Type \to Subst$

$\mathcal{H}(\Gamma, (), \rho) = \mathcal{U}(\rho, \iota)$ ($\mathcal{H}$.1)

$\mathcal{H}(\Gamma, x, \rho) = \mathcal{U}(\rho, \{\vec{\beta}/\vec{\alpha}\}\tau)$ where $\Gamma(x) = \forall\vec{\alpha}.\tau$, new $\vec{\beta}$ ($\mathcal{H}$.2)

$\mathcal{H}(\Gamma, \lambda x.e, \rho) =$ let $S_1 = \mathcal{U}(\rho, \beta_1 \to \beta_2)$, new β_1, β_2 ($\mathcal{H}$.3)
$S_2 = \mathcal{H}(S_1\Gamma + x: S_1\beta_1, e, S_1\beta_2)$ ($\mathcal{H}$.4)
in S_2S_1

$\mathcal{H}(\Gamma, e_1\ e_2, \rho) =$ let $S_1 = \mathcal{H}(\Gamma, e_1, \beta_1 \to \beta_2)$, new β_1, β_2 ($\mathcal{H}$.5)
$S_2 = \mathcal{U}(S_1\rho, S_1\beta_2)$ ($\mathcal{H}$.6)
$S_3 = \mathcal{H}(S_2S_1\Gamma, e_2, S_2S_1\beta_1)$ ($\mathcal{H}$.7)
in $S_3S_2S_1$

$\mathcal{H}(\Gamma, \texttt{let}\ x = e_1\ \texttt{in}\ e_2, \rho) =$
let $S_1 = \mathcal{H}(\Gamma, e_1, \beta)$, new β ($\mathcal{H}$.8)
$S_2 = \mathcal{H}(S_1\Gamma + x: Clos_{S_1\Gamma}(S_1\beta), e_2, S_1\rho)$ ($\mathcal{H}$.9)
in S_2S_1

$\mathcal{H}(\Gamma, \texttt{fix}\ f\ \lambda x.e, \rho) = \mathcal{H}(\Gamma + f: \rho, \lambda x.e, \rho)$ ($\mathcal{H}$.10)

그림 6 H의 정의. 새로 만들어지는 타입 변수들은 서로 다르다. 그리고, 각 $H(\Gamma, e, \rho)$의 재귀 호출에서 만들어지는 타입 변수들의 집합 V는 $V \cap (ftv(\Gamma) \cup ftv(\rho)) = \emptyset$을 만족한다.

$\beta_1 \to \beta_2$로 강요한다. 이렇게 H는 M과는 달리 함수 적용 전체에 강요된 타입 ρ를 전달하지 않는다.

H는 M보다 늦게 오류를 감지한다. H는 함수부분에 덜 구체적인 타입으로 강요하므로 만들어진 구체적인 타입을 늦게 검사하기 때문이다. (H.6)을 보면,

$$S_2 = \mathcal{U}(S_1\rho, S_1\beta_2) \quad \text{(H.6)}$$

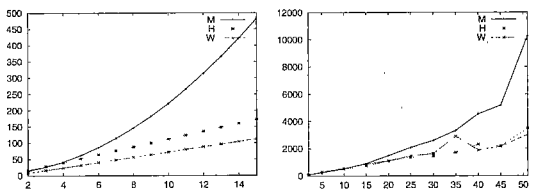
e_1의 구체적인 결과 타입인 $S_1\beta_2$를 현재 강요된 타입 ρ에 비추어 검사한다.

여전히 H는 W보다 일찍 오류를 감지한다. M과 같이, H는 함수 부분에서 만들어진 타입을 인자 부분에 전달하여 이를 상수와 변수까지 흘려보내기 때문이다. (H.7)을 보면,

$$S_3 = \mathcal{H}(S_2S_1\Gamma, e_2, S_2S_1\beta_1) \quad \text{(H.7)}$$

함수 부분 e_1의 인자 타입 $S_2S_1\beta_1$으로 인자 부분 e_2를 강요한다.

항등 함수를 중첩 적용한 프로그램을 가지고 실험한 결과(그림 7)를 볼 때, H에서는 M의 병목현상이 일어나지 않는다. H에서 실행비용은 W와 같이 항등 함수의 개수에 대해 $O(n)$으로 증가하는 것을 볼 수 있다.



(a) 타입크기 (b) 실행시간 (μ-sec)

그림 7 H에서는 M의 병목현상이 일어나지 않는다. (X: 항등 함수의 개수, Y: 측정치)

5. H의 성질

H 알고리즘은 Hindely/Milner let-다형성 타입체계를 충실히 즉, 안전(sound)하고 완전(complete)하게 구현한다. 안전성(soundness)이란 타입 유추 알고리즘이 프로그램의 타입을 유추한 결과가 타입 유추 규칙에 맞다는 것이고, 완전성(completeness)이란 타입 유추 규칙에 맞는 타입을 타입 유추 알고리즘이 유추해 낼 수 있다는 것이다. 안전성과 완전성이 보장되는 타입 유추 알고리즘이란, 그 타입 유추 규칙을 충실히 즉, 제대로 구현한 알고리즘이란 뜻이다.

정리 2 (H의 안정성). *프로그램 e, 타입 환경 Γ, 타입 ρ에 대해, $H(\Gamma, e, \rho)$가 성공하여 결과 S를 주면, $S\Gamma \vdash e: S\rho$가 성립한다.*

증명. 다음 보조정리를 사용하여 증명한다:

보조정리 1[2]. *$\Gamma \vdash e: \tau$이면, $S\Gamma \vdash e: S\tau$가 성립한다.*

e의 구조에 대한 귀납법으로 증명한다. 여기서는 함수 적용의 경우만 증명한다. 나머지 경우는 M의 안전성 증명[3]과 같다.

귀납 가정(induction hypothesis)에 의해 (H.5)로부터 $S_1\Gamma \vdash e_1 : S_1(\beta_1 \to \beta_2)$가 성립한다. 보조정리 1에 의해 양변에 S_3S_2를 적용하면

$$S_3S_2S_1\Gamma \vdash e_1 : S_3S_2S_1(\beta_1 \to \beta_2) \quad (1)$$

가 성립한다. (H.6)에 의해 $S_2S_1\beta_2 = S_2S_1\rho$가 성립하므로, (1)로부터

$$S_3S_2S_1\Gamma \vdash e_1 : S_3S_2S_1(\beta_1 \to \rho) \quad (2)$$

가 성립한다. 귀납 가정에 의해 (H.7)로부터

$$S_3S_2S_1\Gamma \vdash e_2 : S_3S_2S_1\beta_1 \quad (3)$$

가 성립한다. (APP) 규칙에 의해 (2)와 (3)으로부터 $S_3S_2S_1\Gamma \vdash e_1\ e_2 : S_3S_2S_1\rho$가 성립한다. □

정리 3 (H의 완전성). *프로그램 e, 타입 환경 Γ, 타입 ρ에 대해, $P\Gamma \vdash e: P\rho$이면, $H(\Gamma, e, \rho)$가 성공하고, 그 결과를 S라 하면, $P\restriction_V = (RS)\restriction_V$를 만족하는 타입 치환 함수 R이 존재한다. 여기서 V는 $H(\Gamma, e, \rho)$에서 사용된 새 타입 변수들의 집합이다.*

증명. 이 증명은 다음 보조정리들을 사용한다.

보조정리2. *$S = H(\Gamma, e, \rho)$이면 $itv(S) \subseteq ftv(\Gamma) \cup ftv(\rho) \cup V$이다. 이 때, V는 $H(\Gamma, e, \rho)$에서 사용된 새 타입 변수들의 집합이다.*

증명. Lee와 Yi의 보조정리 6의 증명[3]과 유사하게

증명된다. □

보조정리 3 [3]. *$itv(S) \cap A = \emptyset$ 이면 $(RS)\restriction_A = R\restriction_A S$ 이다.*

e의 구조에 대해 귀납법으로 증명한다. 안전성 증명과 마찬가지로 함수 적용의 경우에 대해서만 증명한다.

주어진 가정이 $P\Gamma \vdash e_1\ e_2 : P\rho$ 이고 새 타입 변수들의 집합 $V = \{\beta_1, \beta_2\} \cup V_1 \cup V_2$ 이라고 하자. 여기서 β_1과 β_2는 (H.5)의 새 타입 변수를 가리키고, V_1는 (H.5)의 $H(\Gamma, e_1, \beta_1 \to \beta_2)$에서, 그리고 V_2는 (H.7)의 $H(S_2S_1\Gamma, e_2, S_2S_1\beta_1)$에서 쓰인 새 타입 변수들의 집합이다.

(APP) 규칙에 의해 다음을 만족하는 타입 τ가 존재한다:

$$P\Gamma \vdash e_1 : \tau \to P\rho, \tag{4}$$

$$P\Gamma \vdash e_2 : \tau. \tag{5}$$

$P' = \{\tau/\beta_1, P\rho/\beta_2\} \cup P\restriction_{\{\beta_1,\beta_2\}}$라 두면, $\beta_1, \beta_2 \notin ftv(\Gamma)$이기 때문에 $\tau \to P\rho = P'(\beta_1 \to \beta_2)$이고 $P\Gamma = P'\Gamma$이다. 그러므로, 귀납 가정에 의해 (4)와 (H.5)로부터

$$P'\restriction_{V_1} = (R_1S_1)\restriction_{V_1} \tag{6}$$

을 만족하는 타입 치환 함수 R_1이 존재한다.

$R_1S_1\beta_2 = R_1S_1\rho$ 이다. 왜냐하면,

$$\begin{aligned} R_1S_1\beta_2 &= P'\beta_2 && \beta_2 \notin V_1\text{이므로 (6)에 의해} \\ &= P\rho && P'\text{의 정의로부터} \\ &= P'\rho && \beta_1, \beta_2 \notin ftv(\rho)\text{ 이므로} \\ &= R_1S_1\rho && V_1 \cap ftv(\rho) = \emptyset\text{이므로 (6)에 의해.} \end{aligned}$$

그러므로, (H.6)의 동일화 $U(S_1\beta_2, S_1\rho)$는 성공한다. 그 결과를 S_2라 하면 $R_2S_2 = R_1$를 만족하는 R_2가 존재한다. 그러면, (6)으로부터 다음이 만족된다

$$P'\restriction_{V_1} = (R_2S_2S_1)\restriction_{V_1} \tag{7}$$

(5)와 (H.7)의 $H(S_2S_1\Gamma, e_2, S_2S_1\beta_1)$로부터 귀납 가정에 의해

$$R_2\restriction_{V_2} = (R_3S_3)\restriction_{V_2} \tag{8}$$

를 만족하는 타입 치환 함수 R_3가 존재한다. 왜냐하면,

$$\begin{aligned} \tau &= P'\beta_1 && P'\text{의 정의에 의해서} \\ &= R_2S_2S_1\beta_1 && \beta_1 \notin V_1\text{ 이므로 (7)에 의해.} \end{aligned}$$

그리고

$$\begin{aligned} P\Gamma &= P'\Gamma && \beta_1, \beta_2 \notin ftv(\Gamma)\text{이므로} \\ &= R_2S_2S_1\Gamma && V_1 \cap ftv(\Gamma) = \emptyset\text{ 이므로 (7)에 의해} \end{aligned}$$

이제 끝으로 $P\restriction_V = (R'S_3S_2S_1)\restriction_V$를 만족하는 타입 치환 함수 R'이 존재함을 보이면 된다. $R' = R_3$라고 하면,

$$(R_3S_3S_2S_1)\restriction_V = ((R_3S_3S_2S_1)\restriction_{V_2})\restriction_{V_1\cup\{\beta_1,\beta_2\}} \tag{9}$$

가 성립한다. 여기서 $(itv(S_1) \cup itv(S_2)) \cap V_2 = \emptyset$ 임을 주목하라. 왜냐하면, 정리 1과 보조정리 2에 의해 $itv(S_1) \cup itv(S_2) \subseteq ftv(\Gamma) \cup \{\beta_1, \beta_2\} \cup V_1 \cup ftv(\rho)$이고, 모든 새로 만들어진 타입 변수들은 서로 다르다고 가정했으므로 $(ftv(\Gamma) \cup ftv(\rho)) \cap V_2 \subseteq (ftv(\Gamma) \cup ftv(\rho)) \cap V = \emptyset$ 이고 $(V_1 \cup \{\beta_1, \beta_2\}) \cap V_2 = \emptyset$이기 때문이다. 결국, 보조정리 3으로부터 (9)는 다음과 같다.

$$\begin{aligned} &(R_3S_3S_2S_1)\restriction_V \\ &= ((R_3S_3)\restriction_{V_2})S_2S_1)\restriction_{V_1\cup\{\beta_1,\beta_2\}} \\ &= (R_2\restriction_{V_2}S_2S_1)\restriction_{V_1\cup\{\beta_1,\beta_2\}} && (8)\text{에 의해서} \\ &= (R_2S_2S_1)\restriction_{V_1\cup V_2\cup\{\beta_1,\beta_2\}} && \text{보조정리 3으로부터} \\ &= ((R_2S_2S_1)\restriction_{V_1})\restriction_{V_2\cup\{\beta_1,\beta_2\}} \\ &= P'\restriction_V && (7)\text{에 의해서} \\ &= P\restriction_V && \beta_1, \beta_2 \in V\text{이기 때문에.} \end{aligned}$$

□

H 알고리즘은 W 알고리즘보다 일찍, M 알고리즘보다는 늦게 오류를 감지한다. 호출 문자열(call string) [3]을 사용하여 이를 엄밀하게 살펴보자. $\mathcal{W}(\Gamma, e)$의 호출 문자열 $[\![\mathcal{W}(\Gamma, e)]\!]$는 공백문자열로부터 시작해서 $W(\Gamma_1, e_1)$이 호출될 때마다 $(\Gamma_1, e_1)^d$의 정보를, 그리고 유추를 끝내고 돌아갈 때 $(\Gamma_1, e_1)^u$의 정보를 붙여 만든 문자열이다. 타입 유추 알고리즘이 동일화 실패로 인해 멈추었을 때, "W가 e에서 실패했다"는 것은 다음과 같이 정의된다.

정의 1 [3]. *타입 환경 Γ과 프로그램 e에 대해, "$W(\Gamma, e)$가 e'에서 실패했다"는 것은 호출 문자열 $[\![\mathcal{W}(\Gamma, e)]\!]$에서 짝이 맞지 않는 정보 중에 오른쪽 끝에 있는 것이 $(\Gamma', e')^d$라는 뜻이다.*

이와 유사하게 M과 H의 호출 문자열은 정의된다. H의 호출 문자열은 W보다 길지 않고 M보다 짧지 않다. 그림 8의 예를 보면, W는 e_2를 성공적으로 타입 유추하고 나서 실패하고, M은 e_2의 not을 유추하던 도중에 실패하고, H는 not을 성공적으로 타입 유추하고 나서 실패한다.

정리 4. *타입 환경 Γ, 프로그램 e, 새 타입 변수 β에 대해,*

$$|[\![\mathcal{M}(\Gamma, e, \beta)]\!]| \le |[\![\mathcal{H}(\Gamma, e, \beta)]\!]| \le |[\![\mathcal{W}(\Gamma, e)]\!]|$$

이다. 이 때, $|s|$는 호출 문자열의 길이이다.

증명. e에 오류가 없으면 호출 문자열의 길이가 같다. e에 오류가 있을 때를 생각해 보자. 이 증명은 다음의 보조정리들을 사용한다.

보조정리 4. *프로그램 e, 타입 환경 Γ, 새 타입 변수 β에 대해, $[\![\mathcal{W}(\Gamma,e)]\!]$에 $(\Gamma^W,e')^d$가 있고 $[\![\mathcal{H}(\Gamma,e,\beta)]\!]$에 $(\Gamma^H,e',\rho^H)^d$가 있을 때, $W(\Gamma^W,e')$가 실패하면 $H(\Gamma^H,e',\rho^H)$도 실패한다.*

증명. 부록 A에 있음. □

보조정리 5. *프로그램 e, 타입 환경 Γ, 새 타입 변수 β에 대해, $[\![\mathcal{M}(\Gamma,e,\beta)]\!]$에 $(\Gamma^M,e',\rho^M)^{d/u}$가 있으면, $[\![\mathcal{H}(\Gamma,e,\beta)]\!]$에 $(\Gamma^H,e',\rho^H)^{d/u}$가 있으며 $R\Gamma^H>\Gamma^M$와 $R\rho^H=\rho^M$를 만족하는 타입 치환 함수 R이 존재한다.*

증명. 부록 B에 있음. □

보조정리 5는 $[\![\mathcal{M}(\Gamma,e,\beta)]\!]\le[\![\mathcal{H}(\Gamma,e,\beta)]\!]$을 직접적으로 증명한다. 왜냐하면, H와 M의 호출 문자열의 순서는 같은데 M에 있으면 해당하는 호출 또는 귀환이 H에 있다는 것은 H의 호출 문자열이 항상 더 길거나 같다는 것을 의미하기 때문이다.

이제 $[\![\mathcal{H}(\Gamma,e,\beta)]\!]\le[\![\mathcal{W}(\Gamma,e)]\!]$을 귀류법으로 증명하겠다. $[\![\mathcal{H}(\Gamma,e,\beta)]\!]>[\![\mathcal{W}(\Gamma,e)]\!]$가 맞다고 가정하자. $W(\Gamma,e)$는 함수 적용 $(e_1\ e_2)$과 재귀 함수정의 $(\mathrm{fix}\ f\ \lambda x.e)$에서만 실패할 수 있다. 즉, (W.6)과 (W.10)에서만 실패 할 수 있다.

- **(W.6)의 경우**: $e_1\ e_2$에서 W가 실패하였다고 하자. 그러면, W와 H의 호출 기록은

$[\![\mathcal{W}(\Gamma,e)]\!]=\cdots(\Gamma_1^W,e_1\ e_2)^d\quad\cdots(\Gamma_2^W,e_2)^u$

$[\![\mathcal{H}(\Gamma,e,\beta)]\!]=\cdots(\Gamma_1^H,e_1\ e_2,\rho_1^H)^d\cdots(\Gamma_2^H,e_2,\rho_2^H)^u(\Gamma_1^H,e_1\ e_2,\rho_1^H)^u\cdots$

와 같다. 왜냐하면, H의 호출 문자열이 W보다 길다고 가정했고, e_2의 타입 유추가 끝난 뒤에 $e_1\ e_2$가 끝나기 때문이다. 이것은 $W(\Gamma_1^W,e_1\ e_2)$가 실패했을 때 $H(\Gamma_1^H,e_1\ e_2,\rho_1^H)$가 성공적으로 유추된다는 것인데, 보조정리 4에 의해 불가능하다. 결국 가정에 모순된다.

- **(W.10)의 경우**: 위의 경우와 비슷하게 가정으로부터 $W(\Gamma^W,\mathrm{fix}f\ \lambda x.e')$가 실패했을 때 $H(\Gamma^H,\mathrm{fix}f\ \lambda x.e',\rho^H)$가 성공한다고 이끌어 낼 수 있다. 그러나, 보조정리 4에 의해 불가능하다. 결국 가정에 모순된다. □

6. 토 론

6.1 Obejective Caml에서의 타입 유추 알고리즘

OCaml 2.02[6]는 H와 유사한 타입 유추 알고리즘을 사용한다. 차이는 함수 적용 부분에 있다. OCaml은 함수 적용에서 함수 부분 e_1의 타입을 함수라고 강요하지 않는다.

$$S_1=\mathcal{M}(\Gamma,\ e_1,\ \beta_1),\ \text{new}\ \beta_1$$

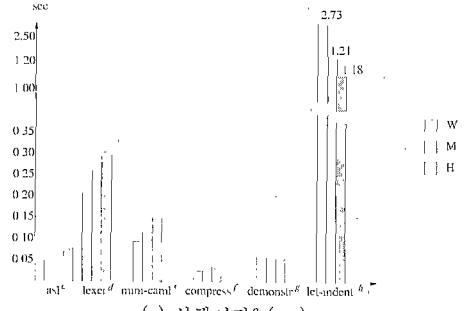
$$\overbrace{\underbrace{(\lambda x.x+1)}_{e_1}\ \underbrace{(\texttt{not true})}_{e_2}}^{e}$$

$[\![\mathcal{W}(\Gamma,\ e)]\!]$ = $(\Gamma,\ e)^d(\Gamma,\ e_1)^d\ldots(\Gamma,\ e_1)^u$
$(\Gamma,\ e_2)^d\ldots(\Gamma,\ e_2)^u$
$\mathcal{W}$는 ($\mathcal{W}$.6)에 의해서 e에서 실패한다.

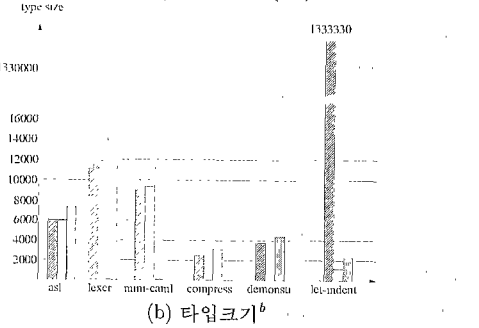
$[\![\mathcal{H}(\Gamma,\ e,\ \beta)]\!]$ = $(\Gamma,\ e,\ \beta)^d(\Gamma,\ e_1,\ \beta_1\to\beta_2)^d\ldots(\Gamma,\ e_1,\ \beta_1\to\beta_2)^u$
$(\Gamma,\ e_2,\ \texttt{int})^d(\Gamma,\ \texttt{not},\ \beta_3\to\beta_4)^d\ldots(\Gamma,\ \texttt{not},\ \beta_3\to\beta_4)^u$
$\mathcal{H}$는 ($\mathcal{H}$.6)에 의해서 e_2에서 실패한다.

$[\![\mathcal{M}(\Gamma,\ e,\ \beta)]\!]$ = $(\Gamma,\ e,\ \beta)^d(\Gamma,\ e_1,\ \beta_1\to\beta)^d\ldots(\Gamma,\ e_1,\ \beta_1\to\beta)^u$
$(\Gamma,\ e_2,\ \texttt{int})^d(\Gamma,\ \texttt{not},\ \beta_2\to\texttt{int})^d$
$\mathcal{M}$은 ($\mathcal{M}$.1)에 의해서 not에서 실패한다.

그림 8 H의 호출 문자열은 W보다 짧고 M보다 길다.



(a) 실행시간[a] (sec)



(b) 타입크기[b]

[a] SUN UltraSparc2 (400MHz)에서 돌림.
[b] 동일화가 처리한 총 타입크기
[c] A Small Language 해석기[8].
[d] Caml 7.4의 어소 분석기(lexer)생성기.
[e] Caml의 축소판 해석기와 타입검사기.
[f] Huffman 화일 압축프로그램.
[g] 항진 명제(tautology) 검사기.
[h] 두번째 최악의 경우(worst case)[9].

그림 9 실험 결과

그리고 함수 부분의 결과 타입을 현재 함수 적용 전체에 강요된 타입 ρ에 비추어 검사를 한다.

$$S_2=\mathcal{U}(S_1\beta_1,\ \beta_2\to S_1\rho),\ \text{new}\ \beta_2$$

그리고 인자 부분 e_2의 타입을 H, M과 같이 강요한다.

$$S_3=\mathcal{M}(S_2S_1\Gamma,\ e_2,\ S_2S_1\beta_1),\ \text{new}\ \beta_1$$

OCaml은 몇몇 예에서 W, M, H와 다르게 오류를 보고한다.

(+ 1 true) 3.

M은 + 에서, H는 true를 유추하기 직전에, OCaml은 true에서, W는 true를 유추한 직후에 실패한다. 즉, OCaml은 H보다 늦게, W보다 일찍 오류를 감지한다.

6.2 실험 결과

일반적인 프로그램을 입력으로 실험한 결과 H 알고리즘은 W, M과 비슷한 성능을 보인다(그림 9). M과 H는 Caml 6.0[4] 컴파일러에, W는 Caml 7.4[7] 컴파일러에 구현되었다. 이 실험은 SUN UltraSparc 2 (400MHz)에서 수행되었다.

let-indent의 경우 W의 결과가 나쁜 이유는 ML의 패턴구조 때문이다. W에서는 let-묶음의 좌변 패턴의 타입과 우변의 타입에 동일화 과정이 필요하다. 반면에, M과 H에서는 이와 같은 과정이 필요 없다. 다음이 let-indent 경우인데:

```
let x0 = fun x -> (x,x) in
let x1 = fun y -> x0 (x0 y) in
let x2 = fun y -> x1 (x1 y) in
x0;;
```

이 경우에서는 프로그램의 크기를 n이라 할 때 타입의 크기가 $O(2^n)$로 증가하는 예이다[9]. 이렇게 거대한 타입을 W는 매번 let-묶음마다 동일화 과정을 해야한다. 그래서 실행비용이 많이 악화된 것이다.

7. 결 론

Hindley/Milner의 let-다형성 타입체계의 표준 타입 유추 알고리즘 W의 오류 보고를 개선하기 위해서 고안된 M은 함수를 중첩으로 사용된 경우 유추에 소요되는 시간이 프로그램 크기의 제곱으로 증가함을 발견하였다.

본 연구에서는 이러한 M의 병목을 해소하기 위해 H 알고리즘을 고안하였다. H는 M 알고리즘이 갖고 있는 장점을 모두 갖고 있음을 증명하였다. 즉, H는 let-다형성 타입체계를 충실히 구현하였고 W 보다 일찍 오류를 감지한다. 또한 M 보다는 오류를 늦게 감지하므로 W와 M과는 다른 오류 보고가 가능하다. 실험을 통하여 H 알고리즘은 W 알고리즘과 M 알고리즘에서 발생할 수 있는 병목현상이 없음을 보였다.

부 록

A. 보조정리 4의 증명

보조정리 4의 증명은 Lee와 Yi의 M 알고리즘에 대한 보조정리 11의 증명[3]과 같다. 단지, 사용되는 보조정리 8만 다른 뿐이다. 그러므로, 여기서는 보조정리 8만 증명한다. 증명을 위하여 다음의 보조정리들이 필요하다.

보조정리 6 [2]. $\sigma > \sigma'$ 이면 $S\sigma > S\sigma'$ 이다.

보조정리 7 [2]. $\Gamma' \vdash e:\tau$가 성립하고, $\Gamma > \Gamma'$ 이면, $\Gamma \vdash e:\tau$가 성립한다.

정리 5 (W 알고리즘의 완전성) [2]. 타입 환경 Γ, 프로그램 e에 대해, $\Gamma' \vdash e:\sigma$를 만족하는 Γ의 예(instance) Γ', 타입스킴 σ가 존재하면, $W(\Gamma, e)$는 성공한다. 그 결과를 (S, τ)라 하면 $\Gamma' = RS\Gamma$이고 $RClos_{S\Gamma}(\tau) > \sigma$를 만족하는 타입 치환 함수 R이 존재한다.

보조정리 8. 표현식 e, 타입 환경 Γ, 새 타입 변수 β에 대하여, $[\![\mathcal{W}(\Gamma, e)]\!]$에 $(\Gamma^W, e')^d$가 있고 $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^H, e', \rho^H)^d$가 있으면, $R\Gamma^W > \Gamma^H$를 만족하는 타입 치환 함수 R이 존재한다.

증명. $[\![\mathcal{W}(\Gamma, e)]\!]$와 $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$의 접두 문자열(prefix), $(\Gamma, e)^d \cdots (\Gamma^W, e')^d$과 $(\Gamma, e, \beta)^d \cdots (\Gamma^H, e', \rho^H)^d$의 길이에 대해 귀납적으로 증명한다. 두 알고리즘은 같은 순서로 표현식을 검색하므로 같은 길이의 접두 문자열을 갖는다. 수식 혼동을 막기 위해 두 알고리즘에서 사용되는 기호들은 위첨자(super-scription)로 구별하기로 한다.

• **초기 경우**: 접두 문자열의 길이가 1일 때, 즉, 처음 호출했을 때, Γ^W와 Γ^H는 같다. R을 $\emptyset$로 두면 조건을 만족한다.

함수 적용의 경우를 제외하고는 Lee와 Yi의 보조정리 10의 증명[3]과 같으므로 생략한다.

• **e'이 $(e'\ e_2)$에 있을 경우**: e'을 호출할 때와 $e'\ e_2$를 호출할 때의 타입 환경이 같으므로, 귀납 가정에 의해 조건이 만족한다.

• **e'이 $(e_1\ e')$에 있을 경우**: H 알고리즘의 안전성에 의해 (H.5)로부터 $S_1^H\Gamma^H \vdash e_1 : S_1^H(\beta_1^H \to \beta_2^H)$가 성립함을 알 수 있다. 보조정리 1에 의해 S_2^H을 양변에 적용할 수 있다. 그러면,

$$S_2^{\mathcal{H}} S_1^{\mathcal{H}} \Gamma^{\mathcal{H}} \vdash e_1 : S_2^{\mathcal{H}} S_1^{\mathcal{H}} (\beta_1^{\mathcal{H}} \to \beta_2^{\mathcal{H}}) \quad (10)$$

이 성립한다. 귀납 가정에 의해 $R_1\Gamma^W > \Gamma^H$이 성립하고, 보조정리 6에 의해

$$S_2^{\mathcal{H}} S_1^{\mathcal{H}} R_1 \Gamma^{\mathcal{W}} \succ S_2^{\mathcal{H}} S_1^{\mathcal{H}} \Gamma^{\mathcal{H}} \quad (11)$$

가 성립한다. 보조정리 7에 의해 (10)은

$$S_2^{\mathcal{H}}S_1^{\mathcal{H}}R_1\Gamma^{\mathcal{W}} \vdash e_1 : S_2^{\mathcal{H}}S_1^{\mathcal{H}}(\beta_1^{\mathcal{H}} \to \beta_2^{\mathcal{H}})$$

을 뜻하고, W 알고리즘의 완전성에 의해 $R'S_1^{\mathcal{W}}\Gamma^{\mathcal{W}} = S_2^{II}S_1^{II}R_1\Gamma^{\mathcal{W}}$을 만족하는 R'이 존재한다. 그러면, (11)에 의해 다음이 성립한다.

$$R'S_1^{\mathcal{W}}\Gamma^{\mathcal{W}} = S_2^{\mathcal{H}}S_1^{\mathcal{H}}R_1\Gamma^{\mathcal{W}} \succ S_2^{\mathcal{H}}S_1^{\mathcal{H}}\Gamma^{\mathcal{H}}. \ \square$$

B. 보조정리 5의 증명

다음의 보조정리들과 M의 안전성이 증명에 사용된다.

보조정리 9 [3]. *타입 치환 함수 S, 타입 환경 Γ, 타입 τ에 대하여, $SClos_{\Gamma}(\tau) > Clos_{S\Gamma}(S\tau)$이다.*

보조정리 10 [3]. *타입 환경 Γ, Γ', 타입 τ에 대하여 $\Gamma > \Gamma'$이면, $Clos_{\Gamma}(\tau) > Clos_{\Gamma'}(\tau)$이다.*

정리 6 (M 알고리즘의 안전성) [3]. *표현식 e, 타입 환경 Γ에 대하여, $M(\Gamma, e, \rho) = S$인 타입 ρ가 존재하면 $S\Gamma \vdash e : S\rho$가 성립한다.*

$[\![\mathcal{M}(\Gamma, e, \beta)]\!]$의 접두 문자열의 길이에 대한 귀납법으로 증명된다.

• **초기 경우**: 접두 문자열 길이가 1일 때, 즉, 처음 호출했을 때는 $\Gamma^{II} = \Gamma^{M}$이고 $\rho^{II} = \rho^{M}$이다. 고로, R을 $\emptyset$로 두면 조건을 만족한다.

먼저, e'에 대한 유추를 끝냈을 때, 즉, 접두 문자열이 $(\Gamma, e, \beta)\cdots(\Gamma^{M}, e', \rho^{M})^{u}$인 경우에 대해 증명한다.

• **e'으로부터 귀환한 경우**: 접두 문자열이 어떤 Γ^{M}, ρ^{M}에 대해

$$\cdots(\Gamma^{\mathcal{M}}, e', \rho^{\mathcal{M}})^{d}\cdots(\Gamma^{\mathcal{M}}, e', \rho^{\mathcal{M}})^{u}$$

인 경우이다. 접두 문자열에 $(\Gamma^{M}, e', \rho^{M})^{u}$이 있다는 것은 $M(\Gamma^{M}, e', \rho^{M})$이 성공한다는 것을 뜻한다. $M(\Gamma^{M}, e', \rho^{M})$이 성공적으로 유추되어 결과로 S^{M}를 낸다 하면, M 알고리즘의 안전성에 의해

$$S^{\mathcal{M}}\Gamma^{\mathcal{M}} \vdash e' : S^{\mathcal{M}}\rho^{\mathcal{M}}$$

가 성립한다. 접두 문자열에 $(\Gamma^{M}, e', \rho^{M})^{d}$가 있으므로, 귀납 가정에 의해, $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{H}, e', \rho^{H})^{d}$가 있고, 또한, $R\rho^{II} = \rho^{M}$와 $R\Gamma^{II} > \Gamma^{M}$을 만족하는 R이 존재한다. $R\Gamma^{II} > \Gamma^{M}$가 성립하므로 보조정리 6에 의해 $S^{M}R\Gamma^{II} > S^{M}\Gamma^{M}$가 성립한다. 그러면, 보조정리 7에 의해

$$S^{\mathcal{M}}R\Gamma^{\mathcal{H}} \vdash e' : S^{\mathcal{M}}R\rho^{\mathcal{H}}$$

가 성립한다. 결국, H의 완전성에 의해 $H(\Gamma^{II}, e', \rho^{II})$가 성공한다. 이는 즉, $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{II}, e', \rho^{II})^{u}$이 있음을 말한다.

이제 e'을 호출하는 경우, 즉, 접두 문자열이 $(\Gamma, e, \beta)\cdots(\Gamma^{M}, e', \rho^{M})^{d}$인 경우를 증명하자. e'의 위치에 따라 여러 경우로 나누어 증명한다.

• **$\lambda x.e'$의 e'를 호출하는 경우**: 접두 문자열이 어떤 Γ^{M}, ρ^{M}에 대해

$$\cdots(\Gamma^{\mathcal{M}}, \lambda x.e', \rho^{\mathcal{M}})^{d}(\cdots, e', \cdots)^{d}$$

인 경우이다. 접두 문자열에 $(\Gamma^{M}, \lambda x.e', \rho^{M})^{d}$이 있으므로, 귀납 가정에 의해 $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{II}, \lambda x.e', \rho^{II})^{d}$가 있고, 또한,

$$R\rho^{\mathcal{H}} = \rho^{\mathcal{M}} \tag{12}$$

와 $R\Gamma^{II} > \Gamma^{M}$을 만족하는 R이 존재한다. 이는 즉, 보조정리 6에 의해

$$S_1^{\mathcal{M}}R\Gamma^{\mathcal{H}} \succ S_1^{\mathcal{M}}\Gamma^{\mathcal{M}} \tag{13}$$

를 의미한다. R'을 $R|_{\{\beta_1^{\mathcal{H}}, \beta_2^{\mathcal{H}}\}} \cup \{\beta_1^{\mathcal{M}}/\beta_1^{\mathcal{H}}, \beta_2^{\mathcal{M}}/\beta_2^{\mathcal{H}}\}$라 하자. 그러면, $S_1^{M}R'$는 ρ^{II}와 $\beta_1^{II} \to \beta_2^{II}$를 동일화한다. 왜냐하면,

$$\begin{aligned}
S_1^{\mathcal{M}}R'\rho^{\mathcal{H}} &= S_1^{\mathcal{M}}R\rho^{\mathcal{H}} && \text{왜냐하면 } \beta_1^{\mathcal{H}}, \beta_2^{\mathcal{H}} \notin ftv(\rho^{\mathcal{H}}) \\
&= S_1^{\mathcal{M}}\rho^{\mathcal{M}} && (12)\text{에 의해} \\
&= S_1^{\mathcal{M}}(\beta_1^{\mathcal{M}} \to \beta_2^{\mathcal{M}}) && \text{왜냐하면 } S_1^{\mathcal{M}} = \mathcal{U}(\rho^{\mathcal{M}}, \beta_1^{\mathcal{M}} \to \beta_2^{\mathcal{M}}) \\
&= S_1^{\mathcal{M}}R'(\beta_1^{\mathcal{H}} \to \beta_2^{\mathcal{H}}) && R' \text{의 정의에 따라.}
\end{aligned}$$

고로, 동일화에 성공하고 $S_1^{M}R' = R_1S_1^{II}$를 만족하는 R_1이 존재한다. 그러면,

$$\begin{aligned}
&R_1(S_1^{\mathcal{H}}\Gamma^{\mathcal{H}} + x : S_1^{\mathcal{H}}\beta_1^{\mathcal{H}}) \\
&= S_1^{\mathcal{M}}R'(\Gamma^{\mathcal{H}} + x : \beta_1^{\mathcal{H}}) \\
&= S_1^{\mathcal{M}}R\Gamma^{\mathcal{H}} + x : S_1^{\mathcal{M}}\beta_1^{\mathcal{M}} && \text{왜냐하면 } \beta_1^{\mathcal{H}}, \beta_2^{\mathcal{H}} \notin ftv(\Gamma^{\mathcal{H}}) \\
&\succ S_1^{\mathcal{M}}\Gamma^{\mathcal{M}} + x : S_1^{\mathcal{M}}\beta_1^{\mathcal{M}} && (13)\text{에 의해}
\end{aligned}$$

가 성립하고, 또한

$$R_1S_1^{\mathcal{H}}\beta_2^{\mathcal{H}} = S_1^{\mathcal{M}}R'\beta_2^{\mathcal{H}} = S_1^{\mathcal{M}}\beta_2^{\mathcal{M}}$$

가 성립한다.

• **e' e_2의 e'를 호출한 경우**: 접두 문자열이

$$\cdots(\Gamma^{\mathcal{M}}, e'\ e_2, \rho^{\mathcal{M}})^{d}(\Gamma^{\mathcal{M}}, e', \beta^{\mathcal{M}} \to \rho^{\mathcal{M}})^{d}$$

인 경우이다. 접두 문자열에 $(\Gamma^{M}, e'\ e_2, \rho^{M})^{d}$가 있으므로, 귀납 가정에 의해 $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{II}, e'\ e_2, \rho^{II})^{d}$가 있고, $R\Gamma^{II} > \Gamma^{M}$와 $R\rho^{II} = \rho^{M}$를 만족하는 R이 존재한다. $e'\ e_2$를 호출한 후 e'를 호출하기까지 동일화 알고리즘을 사용 않으므로, $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{II}, e', \beta_1^{II} \to \beta_2^{II})^{d}$가 있다. R'을 $R|_{\{\beta_1^{\mathcal{H}}, \beta_2^{\mathcal{H}}\}} \cup \{\beta^{\mathcal{M}}/\beta_1^{\mathcal{H}}, \rho^{\mathcal{M}}/\beta_2^{\mathcal{H}}\}$라 하면 $R'(\beta_1^{II} \to \beta_2^{II}) = \beta^{M} \to \rho^{M}$가 성립하고, $\beta_1^{II}, \beta_2^{II} \notin ftv(\Gamma^{II})$이므로 $R'\Gamma^{II} = R\Gamma^{II} > \Gamma^{M}$가 성립한다.

• **e_1 e'의 e'를 호출한 경우**: 접두 문자열이

$$\cdots(\Gamma^{\mathcal{M}}, e_1\ e', \rho^{\mathcal{M}})^{d}\cdots(\Gamma^{\mathcal{M}}, e_1, \beta^{\mathcal{M}} \to \rho^{\mathcal{M}})^{u}(\cdots, e', \cdots)^{d}$$

인 경우이다. 여기서 β^M은 (M.5)에서 생성한 새 타입 변수이다. 접두 문자열에 $(\Gamma^M, e_1, \beta^M \to \rho^M)^u$이 있으므로, $M(\Gamma^M, e_1, \beta^M \to \rho^M)$는 성공한다. M의 안전성에 의해

$$S_1^{\mathcal{M}}\Gamma^{\mathcal{M}} \vdash e_1 : S_1^{\mathcal{M}}(\beta^{\mathcal{M}} \to \rho^{\mathcal{M}}) \tag{14}$$

가 성립한다. 접두 문자열에 $(\Gamma^M, e_1 e', \rho^M)^d$가 있으므로, 귀납 가정에 의해 $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{II}, e_1 e', \rho^{II})^d$가 있고

$$R\rho^{\mathcal{H}} = \rho^{\mathcal{M}} \tag{15}$$

와 $R\Gamma^{II} \succ \Gamma^M$을 만족하는 R이 존재한다. 이는 보조정리 6에 의해

$$S_1^{\mathcal{M}} R\Gamma^{\mathcal{H}} \succ S_1^{\mathcal{M}}\Gamma^{\mathcal{M}} \tag{16}$$

임을 뜻한다. R'을 $R|_{\{\beta_1^{\mathcal{H}}, \beta_2^{\mathcal{H}}\}} \cup \{\beta^{\mathcal{M}}/\beta_1^{\mathcal{H}}, \rho^{\mathcal{M}}/\beta_2^{\mathcal{H}}\}$로 두면, $S_1^M R'(\beta_1^{II} \to \beta_2^{II}) = S_1^M(\beta^M \to \rho^M)$이고 $S_1^M R'\Gamma^{II} = S_1^M R\Gamma^{II} \succ S_1^M \Gamma^M$가 성립한다. 그러므로, 보조정리 7에 의해 (14)로부터

$$S_1^{\mathcal{M}} R'\Gamma^{\mathcal{H}} \vdash e_1 : S_1^{\mathcal{M}} R'(\beta_1^{\mathcal{H}} \to \beta_2^{\mathcal{H}})$$

를 이끌어 낼 수 있다. 그러므로, H의 완전성에 의해 $H(\Gamma^{II}, e_1, \beta_1^{II} \to \rho_2^{II})$에서 발생한 새 타입 변수의 집합을 V라 할 때

$$(R_1 S_1^{\mathcal{H}})|_V = (S_1^{\mathcal{M}} R')|_V \tag{17}$$

을 만족하는 R_1이 존재한다. 바로 이 R_1이 (H.6)의 $S_1^{II}\rho^{II}$와 $S_1^{II}\beta_2^{II}$를 동일화한다. 왜냐하면,

$$\begin{aligned}
R_1 S_1^{\mathcal{H}}\rho^{\mathcal{H}} &= S_1^{\mathcal{M}} R'\rho^{\mathcal{H}} && V \cap ftv(\rho^{\mathcal{H}}) = \emptyset\text{이므로 (17)에 의해}\\
&= S_1^{\mathcal{M}}\rho^{\mathcal{M}} && \text{(15)에 의해}\\
&= S_1^{\mathcal{M}} R'\beta_2^{\mathcal{H}} && R'\text{의 정의에 따라}\\
&= R_1 S_1^{\mathcal{H}}\beta_2^{\mathcal{H}} && \beta_2^{\mathcal{H}} \notin V\text{이므로 (17)에 의해.}
\end{aligned}$$

그러므로, $R_2 S_2^{II} = R_1$인 R_2가 존재하고 (17)로부터

$$(R_2 S_2^{\mathcal{H}} S_1^{\mathcal{H}})|_V = (S_1^{\mathcal{M}} R)|_V \tag{18}$$

를 이끌어 낼 수 있다. 그러면,

$$\begin{aligned}
R_2 S_2^{\mathcal{H}} S_1^{\mathcal{H}}\Gamma^{\mathcal{H}} &= S_1^{\mathcal{M}} R\Gamma^{\mathcal{H}} && V \cap ftv(\Gamma^{\mathcal{H}}) = \emptyset\text{이므로 (18)에 의해}\\
&\succ S_1^{\mathcal{M}}\Gamma^{\mathcal{M}} && \text{(16)에 의해}\\
R_2 S_2^{\mathcal{H}} S_1^{\mathcal{H}}\beta_1^{\mathcal{H}} &= S_1^{\mathcal{M}} R'\beta_1^{\mathcal{H}} && \beta_1^{\mathcal{H}} \notin V_1\text{이므로 (18)에 의해}\\
&= S_1^{\mathcal{M}}\beta_1^{\mathcal{M}} && R'\text{의 정의에 따라}
\end{aligned}$$

이 성립한다.

• let $x = e'$ in e_2 **의 e'를 호출한 경우**: 접두 문자열이 (M.7)에서 생성된 새 타입 변수를 β^M이라 할 때

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

인 경우이다. 접두 문자열에 $(\Gamma^M, \text{let } x = e' \text{ in } e_2, \rho^M)^d$가 있으므로, 귀납 가정에 의해 $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{II}, \text{let } x = e' \text{ in } e_2, \rho^{II})^d$가 있고, $R\Gamma^{II} \succ \Gamma^M$와 $R\rho^{II} = \rho^M$를 만족하는 R이 존재한다. H 알고리즘은 e'를 호출하기까지 동일화 알고리즘을 사용하지 않으므로, $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{II}, e', \beta^{II})^d$가 있다. R'을 $R|_{\{\beta^{\mathcal{H}}\}} \cup \{\beta^{\mathcal{M}}/\beta^{\mathcal{H}}\}$라 두면, $R'\beta^{II} = \beta^M$이고, $\beta^{II} \notin ftv(\Gamma^{II})$이므로 $R'\Gamma^{II} = R\Gamma^{II} \succ \Gamma^M$이다.

• let $x = e_1$ in e' **의 e'를 호출한 경우**: 접두 문자열이 $\cdots(\Gamma^{\mathcal{M}}, \texttt{let } x = e_1 \texttt{ in } e', \rho^{\mathcal{M}})^d \cdots (\Gamma^{\mathcal{M}}, e_1, \beta^{\mathcal{M}})^u(\cdots, e', \cdots)^d$

인 경우이다. 접두 문자열에 $(\Gamma^M, e_1, \beta^M)^u$가 있으므로 $M(\Gamma^M, e_1, \beta^M)$는 성공한다. 그러므로, M 알고리즘의 안전성에 의해

$$S_1^{\mathcal{M}}\Gamma^{\mathcal{M}} \vdash e_1 : S_1^{\mathcal{M}}\beta^{\mathcal{M}} \tag{19}$$

가 성립한다. 접두 문자열에 $(\Gamma^M, \text{let } x = e' \text{ in } e_2, \rho^M)^d$가 있으므로, 귀납 가정에 의해 $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{II}, \text{let } x = e' \text{ in } e_2, \rho^{II})^d$가 있고,

$$R\rho^{\mathcal{H}} = \rho^{\mathcal{M}} \tag{20}$$

와 $R\Gamma^{II} \succ \Gamma^M$을 만족하는 R이 존재한다. 이는 즉, 보조정리 6에 의해

$$S_1^{\mathcal{M}} R\Gamma^{\mathcal{H}} \succ S_1^{\mathcal{M}}\Gamma^{\mathcal{M}} \tag{21}$$

임을 뜻한다. R'을 $R|_{\{\beta^{\mathcal{H}}\}} \cup \{\beta^{\mathcal{M}}/\beta^{\mathcal{H}}\}$라 두면, $S_1^M R'\beta^{II} = S_1^M\beta^M$가 성립하고, $\beta^{II} \notin ftv(\Gamma^{II})$이므로 (21)에 의해 $S_1^M R'\Gamma^{II} = S_1^M R\Gamma^{II} \succ S_1^M\Gamma^M$가 성립한다. 그러면, 보조정리 7에 의해 (19)로부터 $S_1^{\mathcal{M}} R'\Gamma^{\mathcal{H}} \vdash e_1 : S_1^{\mathcal{M}} R'\beta^{\mathcal{H}}$를 이끌어 낼 수 있다. H 알고리즘의 완전성에 의해

$$(R_1 S_1^{\mathcal{H}})|_{V_1} = (S_1^{\mathcal{M}} R')|_{V_1} \tag{22}$$

을 만족하는 R_1이 존재한다. 그러면,

$$\begin{aligned}
R_1 S_1^{\mathcal{H}}\Gamma^{\mathcal{H}} &= S_1^{\mathcal{M}} R'\Gamma^{\mathcal{H}} && V_1 \cap ftv(\Gamma^{\mathcal{H}}) = \emptyset\text{ 이므로 (22)에 의해}\\
&\succ S_1^{\mathcal{M}}\Gamma^{\mathcal{M}} && \text{(21)에 의해}
\end{aligned}$$

이고,

$$\begin{aligned}
&R_1 Clos_{S_1^{\mathcal{H}}\Gamma^{\mathcal{H}}}((S_1^{\mathcal{H}}\beta^{\mathcal{H}}))\\
&\succ Clos_{R_1 S_1^{\mathcal{H}}\Gamma^{\mathcal{H}}}((R_1 S_1^{\mathcal{H}}\beta^{\mathcal{H}})) && \text{보조정리 9에 의해}\\
&\succ Clos_{S_1^{\mathcal{M}}\Gamma^{\mathcal{M}}}((R_1 S_1^{\mathcal{H}}\beta^{\mathcal{H}})) && \text{(23)이므로 보조정리 10에 의해}\\
&= Clos_{S_1^{\mathcal{M}}\Gamma^{\mathcal{M}}}((S_1^{\mathcal{M}} R'\beta^{\mathcal{H}})) && \beta^{\mathcal{H}} \notin V_1\text{이므로 (22)에 의해}\\
&= Clos_{S_1^{\mathcal{M}}\Gamma^{\mathcal{M}}}((S_1^{\mathcal{M}}\beta^{\mathcal{M}})) && R'\text{의 정의에 따라}
\end{aligned}$$

이다. 게다가

$$\begin{aligned}
R_1 S_1^{\mathcal{H}}\rho^{\mathcal{H}} &= S_1^{\mathcal{M}} R'\rho^{\mathcal{H}} && V_1 \cap ftv(\rho^{\mathcal{H}}) = \emptyset\text{이므로 (22)에 의해}\\
&= S_1^{\mathcal{M}}\rho^{\mathcal{M}} && \text{(20)에 의해}
\end{aligned}$$

이다.

• fix $f\ \lambda x.e'$ **에 있는 $\lambda x.e'$를 호출한 경우**: 접두 문자열이

$$\cdots(\Gamma^{\mathcal{M}}, \texttt{fix } f\ \lambda x.e', \rho^{\mathcal{M}})^d(\Gamma^{\mathcal{M}} + x : \rho^{\mathcal{M}}, e', \rho^{\mathcal{M}})^d$$

인 경우이다. 접두 문자열에 $(\Gamma^M, \text{fix } f\ \lambda x.e', \rho^M)^d$가 있으므로, 귀납 가정에 의해 $[\![\mathcal{H}(\Gamma, e, \beta)]\!]$에 $(\Gamma^{II}, \text{fix } f\ \lambda x.e', \rho^{II})^d$가 있고, $R\Gamma^{II} > \Gamma^M$와 $R\rho^{II} = \rho^M$를 만족하는 R이 존재한다. 그러면,

$$R(\Gamma^{\mathcal{H}} + f:\rho^{\mathcal{H}}) = R\Gamma^{\mathcal{H}} + f:\rho^{\mathcal{M}} \succ \Gamma^{\mathcal{M}} + f:\rho^{\mathcal{M}}$$

이 성립한다. □

참 고 문 헌

[1] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer System Science*, 17:348-375, 1978.

[2] 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.

[3] 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.

[4] Xavier Leroy. The Caml Light system, release 0.6. Institut National de Recherche en Informatique et en Automatique, 1993.

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

[6] Xavier Leroy. The Objective Caml system, release 2.02. Institut National de Recherche en Informatique et en Automatique, 1999.

[7] Xavier Leroy. The Caml Light system, release 0.74. Institut National de Recherche en Informatique et en Automatique, 1997.

[8] Michel Mauny. *Functional programming using Caml Light*, January 1995.

[9] Paris C. Kanellakis, Harry G. Mairson, and John C. Mitchell. Unification and ML type reconstruction. In *Proceedings of the 17th Annual ACM Symposium on Principles of Programming Languages*, pages 382--401, 1990.

주 상 현

1998년 한국과학기술원 전산학과 학사(B.S.). 2000년 한국과학기술원 전산학과 석사(M.S.). 2000년 ~ 현재 SK 텔레콤 중앙연구원 연구기획그룹 연구원.

이 욱 세

1995년 한국과학기술원 전산학과 학사(B.C.). 1997년 한국과학기술원 전산학과 석사(M.S.). 1997년 ~ 현재 한국과학기술원 전자전산학과 박사과정 재학중. 관심분야는 ML, 타입 이론, 프로그램 분석.

이 광 근

정보과학회논문지 : 소프트웨어 및 응용 제 27 권 제 3 호 참조