SNU 4541.664A Program Analysis
Spring 2005
Note 16

Prof. Kwangkeun Yi

단순 타입 시스템 Simple Type System

- 어디보자: 타입 추론
- 추론 규칙의 안전성 증명
- 추론 규칙의 구현

어디보자: 타입 추론

- e의 생김새마다 오직 하나의 규칙. 따라서,

$$\begin{aligned} \Gamma \vdash \lambda x.e : \tau \to \tau' &\Leftrightarrow \Gamma + x : \tau \vdash e : \tau' \\ \Gamma \vdash e\ e' : \tau &\Leftrightarrow \Gamma \vdash e : \tau' \to \tau \ \wedge\ \Gamma \vdash e' : \tau' \\ &\vdots \end{aligned}$$

- 주어진 e에 대해서, $\Gamma \vdash e : \tau$인 τ가 여럿가능

$$\frac{\{x : \iota\} \vdash x : \iota}{\vdash \lambda x.x : \iota \to \iota} \qquad \frac{\{x : \iota \to \iota\} \vdash x : \iota \to \iota}{\vdash \lambda x.x : (\iota \to \iota) \to (\iota \to \iota)}$$

$$\frac{\dfrac{\vdots}{\vdash \lambda x.x + 1 : \iota \to \iota} \quad \dfrac{\vdots}{\vdash (\lambda y.y)\ 2 : \iota}}{\vdash (\lambda x.x + 1)\ ((\lambda y.y)\ 2) : \iota}$$

$$\frac{\dfrac{\vdots}{\vdash \lambda x.x + 1 : \iota \to \iota} \quad \dfrac{\dfrac{\vdots}{\vdash \lambda y.y : \iota \to \iota} \quad \dfrac{!}{\vdash \lambda z.z : \iota}}{\vdash (\lambda y.y)\ (\lambda z.z) : \iota}}{\vdash (\lambda x.x + 1)\ ((\lambda y.y)\ (\lambda z.z)) : \iota}$$

$$\frac{\dfrac{\dfrac{\vdots}{\{f : \tau \to \tau'\} \vdash f : \tau \to \tau'} \quad \dfrac{\vdots}{\{f : \tau \to \tau'\} \vdash f : \tau}\ \tau = \tau \to \tau'}{\{f : \tau \to \tau'\} \vdash f\ f : \tau'}}{\vdash \lambda f.f\ f : (\tau \to \tau') \to \tau'}$$

추론 규칙의 안전성 증명: 추론되는 대로 실행된다

증명방법 I

- Progress: 값이 나올 때 까지 문제없이 진행한다.
  $\vdash e : \tau$ 이고 e가 값이 아니면 반드시 $e \rightarrow e'$.
- Subject Reduction: 진행은 타입을 보존한다.
  $\vdash e : \tau$ 이고 $e \rightarrow e'$ 이면 $\vdash e' : \tau$.

치환 {v/x}e

프로그램 실행의 핵심: $(\lambda x.e)\ v \rightarrow \{v/x\}e$

$$
\begin{array}{rcl}
\{v/x\}n & = & n \\
\{v/x\}x & = & v \\
\{v/x\}y & = & y \quad \text{if } y \neq x \\
\{v/x\}(e_1 + e_2) & = & (\{v/x\}e_1) + (\{v/x\}e_2) \\
\{v/x\}(e_1\ e_2) & = & (\{v/x\}e_1)\ (\{v/x\}e_2) \\
\{v/x\}(\lambda y.e) & = & \lambda y.\{v/x\}e \quad \text{if } y \notin \{x\} \cup FV(v)
\end{array}
$$

사실: 묶여있는 변수만 다른 $\lambda x.e$와 $\lambda x'.e'$는 같은 것; 서로 항상 대신할 수 있다.

- 따라서, $\{v/x\}(\lambda y.e)$가 항상 정의 될 수 있는 $\lambda y.e$라고 (즉, $y \notin \{x\} \cup FV(v)$라고) 간주 해도 무방.

타입이 있으면 문제없이 진행

Theorem (Progress)

$\vdash e : \tau$ 이고 e가 값이 아니면 반드시 진행 $e \rightarrow e'$ 한다.

Proof. $\vdash e : \tau$의 증명에 대한 귀납법으로. [1]
$e = e_1\ e_2$인 경우: $\vdash e_1\ e_2 : \tau$이므로 타입추론 규칙에 의해 $\vdash e_1 : \tau' \rightarrow \tau$이고 $\vdash e_2 : \tau'$ 이다. 따라서, 귀납 가정에 의해서,

다른 경우도 마찬가지로 증명. □

[1] "By induction on typing derivation." 증명 규칙이 e의 구조를 따라 귀납하므로, "e의 구조에 대한 귀납법으로"("By structural induction on e")라고 해도 된다.

타입이 있으면 문제없이 진행

Theorem (Progress)

$\vdash e : \tau$ 이고 e가 값이 아니면 반드시 진행 $e \rightarrow e'$ 한다.

Proof. $\vdash e : \tau$의 증명에 대한 귀납법으로. [1]
$e = e_1\ e_2$인 경우: $\vdash e_1\ e_2 : \tau$이므로 타입추론 규칙에 의해 $\vdash e_1 : \tau' \rightarrow \tau$이고 $\vdash e_2 : \tau'$ 이다. 따라서, 귀납 가정에 의해서,

- e_1이 값이 아니면 진행 $e_1 \rightarrow e_1'$ 하고, 이는 곧 프로그램 실행 $\rightarrow$의 정의에 의해 $e_1\ e_2 \rightarrow e_1'\ e_2$과 같다.

다른 경우도 마찬가지로 증명. □

[1] "By induction on typing derivation." 증명 규칙이 e의 구조를 따라 귀납하므로, "e의 구조에 대한 귀납법으로"("By structural induction on e")라고 해도 된다.

타입이 있으면 문제없이 진행

Theorem (Progress)

$\vdash e : \tau$ 이고 e가 값이 아니면 반드시 진행 $e \to e'$ 한다.

Proof. $\vdash e : \tau$의 증명에 대한 귀납법으로. [1]
$e = e_1\ e_2$인 경우: $\vdash e_1\ e_2 : \tau$이므로 타입추론 규칙에 의해 $\vdash e_1 : \tau' \to \tau$이고 $\vdash e_2 : \tau'$ 이다. 따라서, 귀납 가정에 의해서,

- e_1이 값이 아니면 진행 $e_1 \to e_1'$ 하고, 이는 곧 프로그램 실행 $\to$의 정의에 의해 $e_1\ e_2 \to e_1'\ e_2$과 같다.
- 마찬가지로, e_1이 값이고 e_2가 값이 아니라면 진행 $e_2 \to e_2'$ 하고, 이는 곧 프로그램 실행 $\to$의 정의에 의해 $e_1\ e_2 \to e_1\ e_2'$과 같다.

다른 경우도 마찬가지로 증명. □

[1] "By induction on typing derivation." 증명 규칙이 e의 구조를 따라 귀납하므로, "e의 구조에 대한 귀납법으로"("By structural induction on e")라고 해도 된다.

타입이 있으면 문제없이 진행

Theorem (Progress)

$\vdash e : \tau$ 이고 e가 값이 아니면 반드시 진행 $e \to e'$ 한다.

Proof. $\vdash e : \tau$의 증명에 대한 귀납법으로. [1]
$e = e_1\ e_2$인 경우: $\vdash e_1\ e_2 : \tau$이므로 타입추론 규칙에 의해 $\vdash e_1 : \tau' \to \tau$이고 $\vdash e_2 : \tau'$ 이다. 따라서, 귀납 가정에 의해서,

- e_1이 값이 아니면 진행 $e_1 \to e_1'$ 하고, 이는 곧 프로그램 실행 $\to$의 정의에 의해 $e_1\ e_2 \to e_1'\ e_2$과 같다.
- 마찬가지로, e_1이 값이고 e_2가 값이 아니라면 진행 $e_2 \to e_2'$ 하고, 이는 곧 프로그램 실행 $\to$의 정의에 의해 $e_1\ e_2 \to e_1\ e_2'$과 같다.
- e_1과 e_2가 모두 값이라면, $\vdash e_1 : \tau' \to \tau$일 수 있는 값 e_1은 오직 $\lambda x.e'$경우 뿐이다. 따라서 프로그램 실행 $\to$의 정의에 의해 반드시 진행 $e_1\ e_2 = \lambda x.e'\ e_2 \to \{v/x\}e'$한다.

다른 경우도 마찬가지로 증명. □

[1] "By induction on typing derivation." 증명 규칙이 e의 구조를 따라 귀납하므로, "e의 구조에 대한 귀납법으로"("By structural induction on e")라고 해도 된다.

실행은 타입을 보존

Theorem (Subject Reduction, Preservation)

$\vdash e : \tau$ 이고 $e \rightarrow e'$ 이면 $\vdash e' : \tau$.

Proof. $\vdash e : \tau$의 증명에 대한 귀납법으로 진행한다. [2]
$e = e_1\ e_2$인 경우: $\vdash e_1\ e_2 : \tau$이므로 타입추론 규칙에 의해 $\vdash e_1 : \tau' \rightarrow \tau$이고 $\vdash e_2 : \tau'$ 이다. $e_1\ e_2 \rightarrow e'$이라면 세가지 경우밖에 없다:

□

[2] "By induction on typing derivation." 증명 규칙이 e의 구조를 따라 귀납하므로, "e의 구조에 대한 귀납법으로 진행한다"고 해도 된다.

실행은 타입을 보존

Theorem (Subject Reduction, Preservation)

$\vdash e : \tau$ 이고 $e \to e'$ 이면 $\vdash e' : \tau$.

Proof. $\vdash e : \tau$의 증명에 대한 귀납법으로 진행한다. [2]
$e = e_1\ e_2$인 경우: $\vdash e_1\ e_2 : \tau$이므로 타입추론 규칙에 의해 $\vdash e_1 : \tau' \to \tau$이고 $\vdash e_2 : \tau'$ 이다. $e_1\ e_2 \to e'$이라면 세가지 경우밖에 없다:

- $e_1 \to e_1'$이라서 $e_1\ e_2 \to e_1'\ e_2$ 인 경우. 귀납 가정에 의해 $\vdash e_1' : \tau' \to \tau$. $\vdash e_2 : \tau'$이므로, 타입추론 규칙에 의해 $\vdash e_1'\ e_2 : \tau$.

□

[2] "By induction on typing derivation." 증명 규칙이 e의 구조를 따라 귀납하므로, "e의 구조에 대한 귀납법으로 진행한다"고 해도 된다.

실행은 타입을 보존

Theorem (Subject Reduction, Preservation)

$\vdash e : \tau$ 이고 $e \to e'$ 이면 $\vdash e' : \tau$.

Proof. $\vdash e : \tau$의 증명에 대한 귀납법으로 진행한다. [2]
$e = e_1\ e_2$인 경우: $\vdash e_1\ e_2 : \tau$이므로 타입추론 규칙에 의해 $\vdash e_1 : \tau' \to \tau$이고 $\vdash e_2 : \tau'$ 이다. $e_1\ e_2 \to e'$이라면 세가지 경우밖에 없다:

- $e_1 \to e_1'$이라서 $e_1\ e_2 \to e_1'\ e_2$ 인 경우. 귀납 가정에 의해 $\vdash e_1' : \tau' \to \tau$. $\vdash e_2 : \tau'$이므로, 타입추론 규칙에 의해 $\vdash e_1'\ e_2 : \tau$.
- e_1은 값이고 $e_2 \to e_2'$이라서 $e_1\ e_2 \to e_1\ e_2'$ 인 경우. 위의 경우와 유사.

□

[2] "By induction on typing derivation." 증명 규칙이 e의 구조를 따라 귀납하므로, "e의 구조에 대한 귀납법으로 진행한다"고 해도 된다.

실행은 타입을 보존

Theorem (Subject Reduction, Preservation)

$\vdash e : \tau$ 이고 $e \to e'$ 이면 $\vdash e' : \tau$.

Proof. $\vdash e : \tau$의 증명에 대한 귀납법으로 진행한다. [2]
$e = e_1\ e_2$인 경우: $\vdash e_1\ e_2 : \tau$이므로 타입추론 규칙에 의해 $\vdash e_1 : \tau' \to \tau$이고 $\vdash e_2 : \tau'$ 이다. $e_1\ e_2 \to e'$이라면 세가지 경우밖에 없다:

- $e_1 \to e_1'$이라서 $e_1\ e_2 \to e_1'\ e_2$ 인 경우. 귀납 가정에 의해 $\vdash e_1' : \tau' \to \tau$. $\vdash e_2 : \tau'$이므로, 타입추론 규칙에 의해 $\vdash e_1'\ e_2 : \tau$.
- e_1은 값이고 $e_2 \to e_2'$이라서 $e_1\ e_2 \to e_1\ e_2'$ 인 경우. 위의 경우와 유사.
- e_1과 e_2가 모두 값인 경우. $\vdash e_1 : \tau' \to \tau$인 값 e_1은 타입추론 규칙에 의해 $\lambda x.e'$ 밖에는 없다. 즉, $e_1\ e_2 = (\lambda x.e')\ v$ 이고, $(\lambda x.e')\ v \to \{v/x\}e'$ 이다. $\vdash \lambda x.e' : \tau' \to \tau$이라면 타입추론 규칙에 의햇 $x : \tau' \vdash e' : \tau$이다. $\vdash v : \tau'$ 이므로, "Preservation under Substitution Lemma"에 의해 $\vdash \{v/x\}e' : \tau$이다.

□

[2] "By induction on typing derivation." 증명 규칙이 e의 구조를 따라 귀납하므로, "e의 구조에 대한 귀납법으로 진행한다"고 해도 된다.

치환은 타입을 보존

Theorem (Preservation under Substitution)

$\Gamma \vdash v : \tau'$ 이고 $\Gamma + x : \tau' \vdash e : \tau$ 이면 $\Gamma \vdash \{v/x\}e : \tau$.

Proof. $\Gamma + x : \tau' \vdash e : \tau$의 증명에 대한 귀납법으로 증명한다.
$e = \lambda y.e'$ 인 경우: 항상 $y \notin \{x\} \cup FV v$인 $\lambda y.e'$로 간주할 수 있으므로 $\{v/x\}\lambda y.e' = \lambda y.\{v/x\}e'$. 따라서, 보일 것은 $\Gamma \vdash \lambda y.\{v/x\}e' : \tau \stackrel{\text{let}}{=} \tau_1 \to \tau_2$.
가정 $\Gamma + x : \tau' \vdash \lambda y.e' : \tau_1 \to \tau_2$ 으로부터 타입추론 규칙에 의해 $\Gamma + x : \tau' + y : \tau_1 \vdash e' : \tau_2$ 이고, $\Gamma \vdash v : \tau'$와 $y \notin FV(v)$ 으로부터 $\Gamma + y : \tau_1 \vdash v : \tau'$[3] 이므로, 귀납 가정에 의해 $\Gamma + y : \tau_1 \vdash \{v/x\}e' : \tau_2$. 즉, 타입추론 규칙에 의해 $\Gamma \vdash \lambda y.\{v/x\}e' : \tau_1 \to \tau_2$.
다른 경우는 더욱 단순한 귀납. □

[3] 물타기(*Weakening Lemma*) 정리(정의+증명?)

증명 방법 II

다음 두개만으로도 충분:

- $e \rightarrow error$를 정의
- Subject Reduction:
  $\vdash e : \tau$ 이고 $e \rightarrow e'$ 이면 $\vdash e' : \tau$.

왜?

- 목표: 값이 아닌 식 e가 타입이 있으면, e가 문제없이 진행하며 끝난다면 그 타입의 값이어야.

증명 방법 II

다음 두개만으로도 충분:

- $e \rightarrow error$를 정의
- Subject Reduction:
  $\vdash e : \tau$ 이고 $e \rightarrow e'$ 이면 $\vdash e' : \tau$.

왜?

- 목표: 값이 아닌 식 e가 타입이 있으면, e가 문제없이 진행하며 끝난다면 그 타입의 값이어야.
- 값이 아닌 식 e가 타입이 있다고 하자. 잘 진행하는가? 그렇다, $e \rightarrow error$로 진행할 수 없다. 그렇게 진행한다면 모순: "Subject Reduction Lemma"에 의해서 $error$가 타입이 있어야 하는데 $error$의 타입을 결정하는 규칙은 없다.

증명 방법 II

다음 두개만으로도 충분:

- $e \rightarrow error$를 정의
- Subject Reduction:
  $\vdash e : \tau$ 이고 $e \rightarrow e'$ 이면 $\vdash e' : \tau$.

왜?

- 목표: 값이 아닌 식 e가 타입이 있으면, e가 문제없이 진행하며 끝난다면 그 타입의 값이어야.
- 값이 아닌 식 e가 타입이 있다고 하자. 잘 진행하는가? 그렇다, $e \rightarrow error$로 진행할 수 없다. 그렇게 진행한다면 모순: "Subject Reduction Lemma"에 의해서 $error$가 타입이 있어야 하는데 $error$의 타입을 결정하는 규칙은 없다.
- 그리고 진행이 타입을 항상 보존하므로, 값으로 진행이 끝나게 되면 그 값도 같은 타입.

추론 규칙의 구현

타입에 대한 연립방정식 세우고 풀기

- 타입 연립 방정식 u

$$
\begin{array}{rcll}
u & \rightarrow & \tau \doteq \tau & \text{타입 방정식} \\
 & | & u \wedge u & \text{연립} \\
\tau & \rightarrow & \alpha & \text{타입 변수} \\
 & | & \iota \mid \tau \rightarrow \tau &
\end{array}
$$

타입 연립 방정식 세우기

$$V(\Gamma, e, \tau) = u$$

이랬으면 좋은:

$$S \models V(\Gamma, e, \tau) \quad \Leftrightarrow \quad S\Gamma \vdash e : S\tau.$$

여기서:

$$\text{“}S \models u\text{”} \quad = \quad \text{“}S\text{는 방정식 } u\text{의 해(}\textit{model}\text{)”}$$

$$S \in Subst = TyVar \xrightarrow{\text{fin}} Type \quad \text{이고} \quad Type\text{은 } \tau\text{들의 집합}$$

이며

$$\frac{S\tau_1 = S\tau_2}{S \models \tau_1 \doteq \tau_2} \qquad \frac{S \models u_1 \quad S \models u_2}{S \models u_1 \ \wedge \ u_2}$$

이며

$$\begin{aligned} S\alpha &= \begin{cases} \tau & \text{if } \alpha \mapsto \tau \in S \\ \alpha & \text{if } \alpha \notin \mathsf{Dom}\, S \end{cases} \\ S\iota &= \iota \\ S(\tau \to \tau') &= (S\tau) \to (S\tau') \\ S\Gamma &= \{x : S\tau \mid x : \tau \in \Gamma\} \end{aligned}$$

타입 연립 방정식 세우기 $V(\Gamma, e, \tau)$

$$
\begin{array}{rcll}
V(\Gamma, n, \tau) & = & \tau \doteq \iota & \\
V(\Gamma, x, \tau) & = & \tau \doteq \tau' & \text{if } x : \tau' \in \Gamma \\
V(\Gamma, e_1 + e_2, \tau) & = & \tau \doteq \iota \ \wedge \ V(\Gamma, e_1, \iota) \ \wedge \ V(\Gamma, e_2, \iota) & \\
V(\Gamma, \lambda x.e, \tau) & = & \tau \doteq \alpha_1 \rightarrow \alpha_2 \ \wedge \ V(\Gamma + x : \alpha_1, e, \alpha_2) & \text{new } \alpha_1, \alpha_2 \\
V(\Gamma, e_1 \ e_2, \tau) & = & V(\Gamma, e_1, \alpha \rightarrow \tau) \ \wedge \ V(\Gamma, e_2, \alpha) & \text{new } \alpha
\end{array}
$$

$V(\Gamma, e, \tau)$는 옳은가?

즉,

$$S \models V(\Gamma, e, \tau) \quad \Leftrightarrow \quad S\Gamma \vdash e : S\tau$$

인가?

Proof. e의 구조에 대한 귀납법으로.

$\lambda x.e$인 경우: $S \models V(\Gamma, \lambda x.e, \tau)$ 은

$$\begin{aligned}
&= && S \models \tau \doteq \alpha_1 \to \alpha_2 \ \wedge \ V(\Gamma + x : \alpha_1, e, \alpha_2) \quad \text{new } \alpha_1, \alpha_2 \\
&\Leftrightarrow && S \models \tau \doteq \alpha_1 \to \alpha_2 \\
& && \wedge\, S \models V(\Gamma + x : \alpha_1, e, \alpha_2) \\
&\Leftrightarrow && S\tau = S\alpha_1 \to S\alpha_2 \\
& && \wedge\, S\Gamma + x : S\alpha_1 \vdash e : S\alpha_2 \quad \text{(귀납가정)} \\
&\Leftrightarrow && S\tau = S\alpha_1 \to S\alpha_2 \\
& && \wedge\, S\Gamma \vdash \lambda x.e : S\alpha_1 \to S\alpha_2 \\
&\Leftrightarrow && S\Gamma \vdash \lambda x.e : S\tau.
\end{aligned}$$

다른 경우도 비슷하게.

연립 방정식의 해 구하기

동일화 알고리즘(*unification algorithm*):

> *"A Machine-Oriented Logic Based on the Resolution Principle"*, *J.A.Robinson*, Journal of ACM, *Vol.12, No.1, pp.23-41, 1965.*

- 타입 방정식들($\tau \doteq \tau'$)과 해 공간($Type$)은 위 논문의 동일화(*unification*) 알고리즘으로 풀 수 있는 클래스
- 알고리즘 $\mathcal{U}$는 $T \models u$인 T 중에서 가장 일반적인 해(*most general unifier*)를 구해준다.
  - $\mathcal{U}(u) \stackrel{\text{let}}{=} S \models u$ 이고
  - $T \models u \Rightarrow T = RS$ 인 R이 있다.

알고리즘 $\mathcal{U}(V(\Gamma, e, \alpha))$

$$\mathcal{U}(u) = \text{unify-all}(u, \emptyset : Subst) \quad : Subst = TyVar \xrightarrow{\text{fin}} Type$$

이고

$$\begin{array}{rcl} \text{unify-all}(\tau \doteq \tau', S) & = & (\text{unify}(\tau, \tau'))S \\ \text{unify-all}(u \wedge u', S) & = & \text{let } T = \text{unify-all}(u, S) \text{ in unify-all}(Tu', T) \end{array}$$

이며, 동일화(*unification*) 알고리즘 $\text{unify}(\tau, \tau') : Subst$ 는

$$\begin{array}{rcl} \text{unify}(\iota, \iota) & = & \emptyset \\ \text{unify}(\alpha, \tau) \text{ or unifty}(\tau, \alpha) & = & \{\alpha \mapsto \tau\} \\ \text{unify}(\tau_1 \to \tau_2, \tau_1' \to \tau_2') & = & \text{let } S = \text{unify}(\tau_1, \tau_1') \\ & & \quad S' = \text{unify}(S\tau_2, S\tau_2') \\ & & \text{in } SS' \\ \text{unify}(\_) & = & \text{fail} \end{array}$$

알고리즘은 충실한(*sound & complete*) 구현

안전(*sound*)

$$\mathcal{U}(V(\Gamma, e, \alpha)) = S \quad \Rightarrow \quad S\Gamma \vdash e : S\alpha$$

완전(*complete*)

$$\left.\begin{array}{l} \mathcal{U}(V(\Gamma, e, \alpha)) = S \\ \wedge\ \Gamma' = RS\Gamma \\ \wedge\ \tau' = RS\alpha \end{array}\right\} \quad \Leftarrow \quad \Gamma' \vdash e : \tau'$$

다른 알고리즘 I: V와 unify를 동시에

$$M : TyEnv \times Exp \times Type \to Subst$$

$$\begin{array}{rcl}
M(\Gamma, n, \tau) & = & \mathsf{unify}(\iota, \tau) \\
M(\Gamma, x, \tau) & = & \mathsf{unify}(\tau, \tau') \quad \text{if } x : \tau' \in \Gamma \\
M(\Gamma, \lambda x.e, \tau) & = & \text{let } S = \mathsf{unify}(\alpha_1 \to \alpha_2, \tau) \quad \text{new } \alpha_1, \alpha_2 \\
 & & \quad S' = M(S\Gamma + x : S\alpha_1, e, S\alpha_2) \\
 & & \text{in } S'S \\
M(\Gamma, e\ e', \tau) & = & \text{let } S = M(\Gamma, e, \alpha \to \tau) \quad \text{new } \alpha \\
 & & \quad S' = M(S\Gamma, e', S\alpha) \\
 & & \text{in } S'S \\
M(\Gamma, e + e', \tau) & = & \text{let } S = \mathsf{unify}(\iota, \tau) \\
 & & \quad S' = M(S\Gamma, e, \iota) \\
 & & \quad S'' = M(S'S\Gamma, e', \iota) \\
 & & \text{in } S''S'S
\end{array}$$

안전(*sound*)
$$M(\Gamma, e, \alpha) = S \quad \Rightarrow \quad S\Gamma \vdash e : S\alpha$$

완전(*complete*)
$$\left.\begin{array}{l} M(\Gamma, e, \alpha) = S \\ \wedge\ \Gamma' = RS\Gamma \\ \wedge\ \tau' = RS\alpha \end{array}\right\} \quad \Leftarrow \quad \Gamma' \vdash e : \tau'$$

다른 알고리즘 II: V와 unify를 동시에

$$W : TyEnv \times Exp \to Type \times Subst$$

$$\begin{array}{rcl}
W(\Gamma, n) & = & (\iota, \emptyset) \\
W(\Gamma, x) & = & (\tau, \emptyset) \quad \text{if } x : \tau \in \Gamma \\
W(\Gamma, \lambda x.e) & = & \text{let } (\tau, S) = W(\Gamma + x : \alpha, e) \quad \text{new } \alpha \\
 & & \text{in } (S\alpha \to \tau, S) \\
W(\Gamma, e\ e') & = & \text{let } (\tau, S) = W(\Gamma, e) \\
 & & \quad (\tau', S') = W(S\Gamma, e') \\
 & & \quad S'' = \text{unify}(\tau' \to \alpha, S'\tau) \quad \text{new } \alpha \\
 & & \text{in } (S''S'S, S''\alpha) \\
W(\Gamma, e + e') & = & \text{let } (\tau, S) = W(\Gamma, e) \\
 & & \quad S' = \text{unify}(\tau, \iota) \\
 & & \quad (\tau', S'') = W(S'S\Gamma, e') \\
 & & \quad S''' = \text{unify}(\tau', \iota) \\
 & & \text{in } (\iota, S'''S''S'S)
\end{array}$$

안전(*sound*) $\quad W(\Gamma, e) = (\tau, S) \Rightarrow S\Gamma \vdash e : \tau$

완전(*complete*) $\quad \left.\begin{array}{l} W(\Gamma, e) = (\tau, S) \\ \wedge\ \Gamma' = RS\Gamma \\ \wedge\ \tau' = R\tau \end{array}\right\} \Leftarrow \Gamma' \vdash e : \tau'$