SNU 프로그래밍언어 특강

(2.2)

이 광근

kwangkeunyi.snu.ac.kr

안전한 실행전의미$_{\text{static semantics}}$의 쓰임새

프로그램 실행미래 예측가능

- 오호: 프로그램 실행전의미$_{\text{static semantics}}$를 –타입이 있는지를– 실행전에 확인할 수 있다면?
- 예측가능: 그 프로그램은 잘돌고, 끝나면 예측한 타입의 값을 내놓을 것이다
- 더군다나: 그 확인이 자동으로 + 현실적인 비용으로 된다면
- “2세대 SW검산 기술”

타입 유추type inference

실제 프로그래밍 언어 시스템에 장착하는 전자동 도구로

- 간단한 타입simple type의 경우
  - 타입유추는 이런식으로
  - 단, 간단한 타입만으론 많은 프로그램을 지원못함
- 여러모양 타입polymorphic type의 경우
  - 그래야, 많은 프로그램을 지원할 수 있다
  - "let-polymorphism" (Hindley-Milner)
    - 제한적인 여러모양 타입 (rank 1, "prenex")
    - 자동유추 ("implicit let-polymophism")
    - 현실적인 비용
    - ML, Haskell의 타입유추 엔진기술

실행전의미: 간단한 타입simple type 경우

용어) 실행전의미static semantics

$\simeq$ 타입규칙typing rule, type inference rule

$\simeq$ 타입시스템type system

$$\begin{array}{rcl} \text{식} \quad E & \to & n \\ & | & x \\ & | & E + E \\ & | & \texttt{fn}\ x\ E \\ & | & E\ E \end{array} \qquad \begin{array}{rcl} \text{타입} \quad \tau & \to & \mathsf{int} \\ & | & \tau \to \tau \end{array}$$

(타입환경type env $\Gamma \in \mathsf{Var} \xrightarrow{\text{fin}} \mathsf{Type}$)

$$\frac{}{\Gamma \vdash n : \mathsf{int}} \qquad \frac{\Gamma \vdash E_1 : \mathsf{int} \quad \Gamma \vdash E_1 : \mathsf{int}}{\Gamma \vdash E_1 + E_2 : \mathsf{int}}$$

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

$$\frac{\Gamma + x : \tau_1 \vdash E : \tau_2}{\Gamma \vdash \texttt{fn}\ x\ E : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash E_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash E_2 : \tau_1}{\Gamma \vdash E_1\ E_2 : \tau_2}$$

타입규칙(실행전의미)의 안전성 증명

예측한대로 실행된다

(증명방법 I)

- 타입이 있으면
- Progress Lemma: 값이 나올 때까지 문제없이 진행한다
  $\vdash E : \tau$ 이고 E가 값이 아니면 반드시 $E \to E'$.
- Preservation Lemma: 진행은 타입을 보존한다
  $\vdash E : \tau$ 이고 $E \to E'$ 이면 $\vdash E' : \tau$.

바꿔치기 $\{v/x\}E$

프로그램 실행의 핵심: $(\texttt{fn}\, x\, E)\, v \to \{v/x\}e$

$$
\begin{array}{rcl}
\{v/x\}n & = & n \\
\{v/x\}x & = & v \\
\{v/x\}y & = & y \quad \text{if } y \neq x \\
\{v/x\}(E_1 + E_2) & = & (\{v/x\}E_1) + (\{v/x\}E_2) \\
\{v/x\}(E_1\ E_2) & = & (\{v/x\}E_1)\ (\{v/x\}E_2) \\
\{v/x\}(\texttt{fn}\, y\, E) & = & \texttt{fn}\, y\, (\{v/x\}E) \quad y \notin \{x\} \cup FV(v)
\end{array}
$$

사실: 묶여있는 변수만 다른 $\texttt{fn}\, x\, E$와 $\texttt{fn}\, x'\, E'$는 같은 것; 서로 항상 대신할 수 있다.

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

타입이 있으면 문제없이 진행

Lemma (Progress)

$\vdash E : \tau$ *이고* E*가 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 $\texttt{fn}\ x\ E'$경우 뿐이다. 따라서 프로그램 실행 $\to$의 정의에 의해 반드시 진행 $E_1\ E_2 = (\texttt{fn}\ x\ E')\ E_2 \to \{E_2/x\}e'$한다.

다른 경우도 마찬가지로 증명.

실행은 타입을 보존

Lemma (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 $\texttt{fn}\ x\ E'$ 밖에는 없다. 즉, $E_1\ E_2 = (\texttt{fn}\ x\ E')\ v$ 이고, $(\texttt{fn}\ x\ E')\ v \to \{v/x\}E'$ 이다. $\vdash \texttt{fn}\ x\ E' : \tau' \to \tau$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.

바꿔치기는 타입을 보존

Lemma (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.

$E = \mathtt{fn}\, y\, E'$ 인 경우: 항상 $y \notin \{x\} \cup FV\, v$인 $\mathtt{fn}\, y\, E'$로 간주할 수 있으므로 $\{v/x\}(\mathtt{fn}\, y\, E') = \mathtt{fn}\, y\, (\{v/x\}E')$. 따라서, 보일 것은 $\Gamma \vdash \mathtt{fn}\, y\, (\{v/x\}E') : \tau \overset{\text{let}}{=} \tau_1 \to \tau_2$.

가정 $\Gamma + x : \tau' \vdash \mathtt{fn}\, y\, E' : \tau_1 \to \tau_2$ 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 $\Gamma \vdash \mathtt{fn}\, y\, (\{v/x\}E') : \tau_1 \to \tau_2$.

다른 경우는 더욱 단순한 귀납. □

자동 타입유추type inference: 단순한타입simple type

타입에 대한 연립방정식 세우고 풀기

- 타입 연립 방정식 u

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

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:

$$\begin{aligned} S &\in \text{바꿔치기 } Subst = TyVar \xrightarrow{\text{fin}} Type \\ S \models u &\stackrel{\text{def}}{=} \text{“}S\text{는 방정식 } u\text{의 해}_{\text{model}}\text{”} \end{aligned}$$

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

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

바꿔치기들은 오른쪽부터

$$S_n \cdots S_0 X \stackrel{\text{def}}{=} S_n \cdots (S_0 X)$$

타입 연립 방정식 세우기 $V(\Gamma, E, \tau)$

$$
\begin{aligned}
V(\Gamma, n, \tau) &= \tau \doteq \iota \\
V(\Gamma, x, \tau) &= \tau \doteq \tau' \qquad \text{if } x : \tau' \in \Gamma \\
V(\Gamma, E_1 + E_2, \tau) &= \tau \doteq \iota \ \wedge\ V(\Gamma, e_1, \iota) \ \wedge\ V(\Gamma, e_2, \iota) \\
V(\Gamma, \mathtt{fn}\, x\, E, \tau) &= \tau \doteq \alpha_1 \to \alpha_2 \ \wedge\ V(\Gamma + x : \alpha_1, e, \alpha_2) \\
&\quad \text{new } \alpha_1, \alpha_2 \\
V(\Gamma, E_1\ E_2, \tau) &= V(\Gamma, e_1, \alpha \to \tau) \ \wedge\ V(\Gamma, e_2, \alpha) \\
&\quad \text{new } \alpha
\end{aligned}
$$

$V(\Gamma, E, \tau)$는 옳은가?

즉,

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

인가?

Proof. E의 구조에 대한 귀납법으로.

$\texttt{fn}\, x\, E$인 경우: $S \models V(\Gamma, \texttt{fn}\, x\, E, \tau)$ 은

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

다른 경우도 비슷하게.

연립 방정식의 해 구하기

동일화 알고리즘unification 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 동일화unification 알고리즘으로 풀 수 있는 클래스
- 알고리즘 $\mathcal{U}$는 $T \models u$인 T 중에서 가장 일반적인 해most general unifier를 구해준다.
  - $(\mathcal{U}(u) \stackrel{\text{let}}{=} S) \models u$ 이고
  - $T \models u$ 이면 $\exists R.T = RS$.

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

$$
\begin{array}{rcl}
\mathcal{U}(u) & \in & Subst = TyVar \xrightarrow{\text{fin}} Type \\
\mathcal{U}(\tau \doteq \tau') & = & \mathsf{unify}(\tau, \tau') \\
\mathcal{U}(u \wedge u') & = & \text{let } S = \mathcal{U}(u) \\
 & & \quad S' = \mathcal{U}(Su') \\
 & & \text{in } S'S
\end{array}
$$

동일화 알고리즘 unify

$$
\begin{array}{rcl}
\mathsf{unify}(\tau, \tau') & \in & Subst \\
\mathsf{unify}(\tau, \tau) & = & \emptyset \\
\mathsf{unify}(\alpha, \tau) \ \vee \ \mathsf{unifty}(\tau, \alpha) & = & \begin{cases} \{\alpha \mapsto \tau\} & \text{if } \alpha \notin \tau \\ \mathsf{fail} & \text{else} \end{cases} \\
\mathsf{unify}(\tau_1 \to \tau_2, \tau_1' \to \tau_2') & = & \text{let } S = \mathsf{unify}(\tau_1, \tau_1') \\
 & & \quad S' = \mathsf{unify}(S\tau_2, S\tau_2') \\
 & & \text{in } S'S \\
\mathsf{unify}(\_) & = & \mathsf{fail}
\end{array}
$$

알고리즘은 충실한sound & complete 구현

안전sound

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

완전complete

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

다른 알고리즘 I: V와 unify를 동시에

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

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

안전$_{\text{sound}}$ $\quad M(\Gamma, E, \alpha) = S \;\Rightarrow\; S\Gamma \vdash E : S\alpha$

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

다른 알고리즘 II: V와 unify를 동시에

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

$$\begin{aligned}
W(\Gamma, n) &= (\iota, \emptyset) \\
W(\Gamma, x) &= (\tau, \emptyset) \quad \text{if } x : \tau \in \Gamma \\
W(\Gamma, \texttt{fn}\, x\, E) &= \text{let } (\tau, S) = W(\Gamma + x : \alpha, E) \quad \text{new } \alpha \\
&\quad \text{in } (S\alpha \to \tau, S) \\
W(\Gamma, E\ E') &= \text{let } (\tau, S) = W(\Gamma, E) \\
&\qquad (\tau', S') = W(S\Gamma, E') \\
&\qquad S'' = \text{unify}(\tau' \to \alpha, S'\tau) \quad \text{new } \alpha \\
&\quad \text{in } (S''\alpha, S''S'S) \\
W(\Gamma, E + E') &= \text{let } (\tau, S) = W(\Gamma, E) \\
&\qquad S' = \text{unify}(\tau, \iota) \\
&\qquad (\tau', S'') = W(S'S\Gamma, E') \\
&\qquad S''' = \text{unify}(\tau', \iota) \\
&\quad \text{in } (\iota, S'''S''S'S)
\end{aligned}$$

안전$_{\text{sound}}$

$$W(\Gamma, E) = (\tau, S) \;\Rightarrow\; S\Gamma \vdash E : \tau$$

완전$_{\text{complete}}$

$$\left.\begin{array}{l} W(\Gamma, E) = (\tau, S) \\ \wedge\ \Gamma' = RS\Gamma \\ \wedge\ \tau' = R\tau \end{array}\right\} \Leftarrow\; \Gamma' \vdash E : \tau'$$

타입규칙 쓸모 향상시키기

간단한타입 시스템simple type system이 "잘 모르겠다"고 하는 경우를 줄여보자.

- 타입 시스템에서는

$$\text{"잘 모르겠다"} = \text{"타입방정식의 해가 없다"}$$

- 여러모양타입 시스템polymorphic type system($\vdash_p$)에서는 "잘 모르겠다"고 하는 경우가 적다, 단순한타입 시스템simple type system($\vdash$) 보다:

$$\Gamma \vdash E : \tau \;\Rightarrow\; \Gamma \vdash_p E : \tau$$

- $\vdash_p$는 $\vdash$의 "conservative extension"이라고 함.

쓸모가 적은 간단한타입 시스템

$$\dfrac{\dfrac{\dfrac{\vdots}{\{f : \tau \to \tau'\} \vdash f : \tau \to \tau'} \quad \dfrac{\vdots}{\{f : \tau \to \tau'\} \vdash f : \tau}\ \tau = \tau \to \tau'}{\{f : \tau \to \tau'\} \vdash f\ f : \tau'}}{\vdash \mathtt{fn}\ f\ (f\ f) : (\tau \to \tau') \to \tau'}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\{f : \tau \to \tau\} \vdash f : \tau \to \tau} \quad \dfrac{\vdots}{\{f : \tau \to \tau\} \vdash f : \tau}\ \tau = \tau \to \tau}{\{f : \tau \to \tau\} \vdash f\ f : \tau \to \tau}}{\vdash \mathtt{fn}\ f\ (f\ f) : (\tau \to \tau) \to (\tau \to \tau)}}{\vdash (\mathtt{fn}\ f\ (f\ f))(\mathtt{fn}\ x\ x) : \tau \to \tau} \quad \vdots$$

여러모양타입polymorphic type을 이용하자

여러모양타입 이용하기: 타입을 일반화type generalization 시키는 작업

$$\forall\alpha.\alpha \to \iota, \quad \forall\alpha_1, \alpha_2.\alpha_1 \to \alpha_2, \quad \cdots$$

그래서

$$\dfrac{\dfrac{\dfrac{\vdots}{\{f : \forall\alpha.\alpha \to \alpha\} \vdash f : (\iota \to \iota) \to (\iota \to \iota)} \quad \dfrac{\vdots}{\{f : \forall\alpha.\alpha \to \alpha\} \vdash f : \iota \to \iota}}{\{f : \forall\alpha.\alpha \to \alpha\} \vdash f\ f : \iota}}{\vdash \mathtt{fn}\ f\ (f\ f) : (\forall\alpha.\alpha \to \alpha) \to \iota}$$

이고

$$\dfrac{\dfrac{\vdots}{\vdash \mathtt{fn}\ f\ (f\ f) : (\forall\alpha.\alpha \to \alpha) \to (\iota \to \iota)} \quad \dfrac{\vdots}{\vdash \mathtt{fn}\ x\ x : \iota \to \iota}}{\vdash (\mathtt{fn}\ f\ (f\ f))(\mathtt{fn}\ x\ x) : \iota \to \iota}$$

하지만 함부로 일반화를 이용하면

- 안전하지 않을 뿐더러
- 충실한 구현이 불가능undecidable:
  - 맘대로인 여러모양타입은 피해야
  - "$\forall$"이 맨 바깥prenex form인 여러모양타입만

함부로 일반화하면 불안전

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\{f : \forall\alpha.\alpha \to \iota\} \vdash f : \iota \to \iota} \quad \cdots}{\{f : \forall\alpha.\alpha \to \iota\} \vdash f\ 1 : \iota} \quad \cdots}{\{f : \forall\alpha.\alpha \to \iota\} \vdash (f\ 1, f\ \mathsf{true}) : \iota \times \iota}}{\vdash \mathtt{fn}\ f\ (f\ 1, f\ \mathsf{true}) : (\forall\alpha.\alpha \to \iota) \to (\iota \times \iota)} \quad \dfrac{\vdots}{\vdash \mathtt{fn}\ x\ x + 1 : \iota \to \iota}}{\vdash (\mathtt{fn}\ f\ (f\ 1, f\ \mathsf{true}))(\mathtt{fn}\ x\ x + 1) : \iota \times \iota}$$

혹은

$$\dfrac{\vdots}{\vdash (\mathtt{fn}\ x\ (\mathtt{let}\ y\ x\ (y\ 1, y\ \mathsf{true})))(\mathtt{fn}\ z\ z + 1) : \iota \times \mathit{bool}}$$

안전한 타입규칙 디자인: let-여러모양타입

Hindley-Milner style let-polymorphism

- 프로그램이 특별히 생긴 경우만 그렇게 정교한 유추가 작동하도록
- 함수가 어디서 무슨 인자로 어떻게 사용되는 지를 알 수 있는 경우 즉,

$$(\texttt{fn}\, x\, \underbrace{\cdots x \cdots x \cdots}_{E})E'$$

즉,

$$\texttt{let}\, x\, E'\, E$$

인 경우만

- 이 경우, E'이 여러모양타입일 수 있는 지 "보수적으로" 확인한 후에, E안에서 x가 어떻게 사용되는지 유추.
- 여러모양타입은 $\forall$이 맨 바깥에만:

$$\iota \to \iota, \forall\alpha.\alpha \to \alpha, \forall\alpha_1\alpha_2.\alpha_1 \to \alpha_2$$

여러모양타입 규칙

- 타입type과 타입틀type scheme

$$\begin{array}{rrcl} \text{타입} & \tau & \to & \iota \mid \tau \to \tau \mid \alpha \\ \text{타입틀} & \sigma & \to & \tau \mid \forall \alpha.\sigma \end{array}$$

  타입틀type scheme은 단순한타입과 여러모양타입을 포함.
  여러모양타입은 $\forall$이 맨 바깥에만(prenex form).

- 추론규칙inference rules은 "$\Gamma \vdash E : \tau$"꼴을 유추하는 규칙들
- 가정들 Γ
  - 변수들의 타입틀type scheme에 대한 가정
  - $x + 1 : \iota$, 가정 $x : \iota$ 아래서.
  - $(f\ 1, f\ \mathsf{true}) : \iota \times bool$, 가정 $f : \forall \alpha.\alpha \to \alpha$ 아래서

$\Gamma \vdash E : \tau$ 타입규칙

$$\frac{}{\Gamma \vdash n : \iota} \qquad \frac{}{\Gamma \vdash x : \tau}\ \sigma \succ \tau, x : \sigma\ \in \Gamma$$

$$\frac{\Gamma \vdash E : \tau \quad \Gamma + x : Gen_\Gamma(\tau) \vdash E' : \tau'}{\Gamma \vdash \texttt{let}\ x\ E\ E' : \tau'}$$

$$\frac{\Gamma \vdash E_1 : \iota \quad \Gamma \vdash E_2 : \iota}{\Gamma \vdash E_1 \texttt{ + } E_2 : \iota}$$

$$\frac{\Gamma \vdash E_1 : \tau' \to \tau \quad \Gamma \vdash E_2 : \tau'}{\Gamma \vdash E_1\ E_2 : \tau} \qquad \frac{\Gamma + x : \tau \vdash E : \tau'}{\Gamma \vdash \texttt{fn}\ x\ E : \tau \to \tau'}$$

$$\begin{aligned} Gen_\Gamma(\tau) &= \forall \alpha_1 \cdots \alpha_n . \tau \qquad \{\alpha_1, \cdots, \alpha_n\} = FTV(\tau) \setminus FTV(\Gamma) \\ \sigma &\succ \tau \qquad \sigma = \forall \alpha_1 \cdots \alpha_n . \tau' \ \wedge\ \tau = \{\tau_i / \alpha_i\}_i \tau \\ FTV(\tau) &= TV(\tau) \\ FTV(\forall \alpha . \sigma) &= FTV\sigma \setminus \{\alpha\} \\ FTV(\Gamma) &= \cup_{x:\sigma \in \Gamma} FTV(\sigma) \end{aligned}$$

타입규칙의 안전성: $Gen_\Gamma(\tau)$

왜 타입 τ를 여러모양타입 $\forall\alpha.\tau$ 으로 만들때 α가 Γ 에 나타나면 제외?

$Gen_\Gamma(\tau) = \forall\alpha_1, \cdots, \alpha_n.\tau$ 여기서 $\{\alpha_1, \cdots, \alpha_n\} = FTV(\tau) \backslash FTV(\Gamma)$

- Γ에 가정$(x : \sigma)$이 첨가 되는 경우는 $\texttt{fn}\, x\, E$ 의 경우
- Γ에 있는 가정을 사용하는 경우? 함수안에서 함수의 인자를 사용할때
- 함수의 인자 타입을 특별히 일반화시킨 후 함수안 타입을 유추? 곤란
- 실행중 전달되는 인자는 여러모양타입 값이 아닐 수 있음

예)

$$\texttt{fn}\, x\, (\texttt{let}\, y\, x\, (y\, 1, y\, \mathsf{true}))$$

이 함수가 그런 특별한(여러모양타입) 함수만 x에 전달될것을 확인해야하는데!

타입규칙(실행전의미)의 안전성 증명

예측한대로 실행된다

(증명방법 I)

- 타입이 있으면
- Progress Lemma: 값이 나올 때까지 문제없이 진행한다
  $\vdash E : \tau$ 이고 E가 값이 아니면 반드시 $E \to E'$.
- Preservation Lemma: 진행은 타입을 보존한다
  $\vdash E : \tau$ 이고 $E \to E'$ 이면 $\vdash E' : \tau$.

자동 타입유추: 여러모양 타입polymorphic type 경우

- 간단한타입simple type 유추 알고리즘 M과 W의 "자연스러운" 확장
- $\mathcal{M}$나 $\mathcal{W}$도 모두 충실한 구현. 예를들어,

안전sound

$$\mathcal{W}(\Gamma, E) = (\tau, S) \quad \Rightarrow \quad S\Gamma \vdash E : \tau$$

완전complete

$$\left.\begin{array}{l} \mathcal{W}(\Gamma, E) = (\tau, S) \\ \wedge\ \Gamma' = RS\Gamma \\ \wedge\ R(Gen_{S\Gamma}(\tau)) \succ \tau' \end{array}\right\} \Leftarrow \quad \Gamma' \vdash E : \tau'$$

참고) "Proofs about a Folklore Let-Polymorphic Type Inference Algorithm", Oukseh Lee and Kwangkeun Yi, *ACM Transactions on Programming Langauges and Systems*

let-여러모양타입 유추 알고리즘 $\mathcal{W}$

$$\mathcal{W} : TyEnv \times Exp \to Type \times (TyVar \xrightarrow{\text{fin}} Type)$$

$$\begin{array}{lcll}
\mathcal{W}(\Gamma, n) & = & (\iota, \emptyset) & \\
\mathcal{W}(\Gamma, x) & = & (\{\alpha_i \mapsto \beta_i\}_{i=1}^{n}\tau, \emptyset) & \text{where } \Gamma(x) = \forall\vec{\alpha}.\tau, \text{ new } \vec{\beta} \\
\mathcal{W}(\Gamma, \texttt{fn}\ x\ E) & = & \text{let} & (S_1, \tau_1) = \mathcal{W}(\Gamma + x\colon \beta, E), \text{ new } \beta \\
 & & \text{in} & (S_1\beta \to \tau_1, S_1) \\
\mathcal{W}(\Gamma, E_1\ E_2) & = & \text{let} & (S_1, \tau_1) = \mathcal{W}(\Gamma, E_1) \\
 & & & (S_2, \tau_2) = \mathcal{W}(S_1\Gamma, E_2) \\
 & & & S_3 = \text{unify}(S_2\tau_1, \tau_2 \to \beta), \text{ new } \beta \\
 & & \text{in} & (S_3\beta, S_3S_2S_1) \\
\mathcal{W}(\Gamma, \texttt{let}\ x\ E_1\ E_2) = & & & \\
 & & \text{let} & (S_1, \tau_1) = \mathcal{W}(\Gamma, E_1) \\
 & & & (S_2, \tau_2) = \mathcal{W}(S_1\Gamma + x : GEN_{S_1\Gamma}(\tau_1), E_2) \\
 & & \text{in} & (\tau_2, S_2S_1)
\end{array}$$

let-여러모양타입 유추 알고리즘 $\mathcal{M}$

$$\mathcal{M} : TyEnv \times Exp \times Type \to (TyVar \xrightarrow{\text{fin}} Type)$$

$$\begin{array}{lcll}
\mathcal{M}(\Gamma, n, \tau) & = & \mathsf{unify}(\tau, \iota) & \\
\mathcal{M}(\Gamma, x, \tau) & = & \mathsf{unify}(\tau, \{\alpha_i \mapsto \beta_i\}_{i=1}^{n}\tau') & \text{where } \Gamma(x) = \forall\vec{\alpha}.\tau',\ \text{new } \vec{\beta} \\
\mathcal{M}(\Gamma, \mathtt{fn}\, x\ E, \tau) & = & \mathsf{let} & S_1 = \mathsf{unify}(\tau, \beta_1 \to \beta_2),\ \text{new } \beta_1, \beta_2 \\
 & & & S_2 = \mathcal{M}(S_1\Gamma + x : S_1\beta_1, E, S_1\beta_2) \\
 & & \mathsf{in} & S_2 S_1 \\
\mathcal{M}(\Gamma, E_1\ E_2, \tau) & = & \mathsf{let} & S_1 = \mathcal{M}(\Gamma, E_1, \beta \to \tau),\ \text{new } \beta \\
 & & & S_2 = \mathcal{M}(S_1\Gamma, E_2, S_1\beta) \\
 & & \mathsf{in} & S_2 S_1 \\
\mathcal{M}(\Gamma, \mathtt{let}\, x\ E_1\ E_2, \tau) = & & & \\
 & & \mathsf{let} & S_1 = \mathcal{M}(\Gamma, E_1, \beta),\ \text{new } \beta \\
 & & & S_2 = \mathcal{M}(S_1\Gamma + x : GEN_{S_1\Gamma}(S_1\beta), E_2, S_1\tau) \\
 & & \mathsf{in} & S_2 S_1
\end{array}$$

$\mathcal{M}, \mathcal{W}$ 알고리즘 성질

$|f| \stackrel{\text{let}}{=}$ 재귀함수 f의 실행 반복횟수

- $|\mathcal{M}| \leq |\mathcal{W}|$: $\mathcal{M}$은 $\mathcal{W}$보다 일찍끝난다(타입없는 프로그램의 경우)
- $|\mathcal{M}|, |\mathcal{W}| \sim |E|$ 입력 프로그램 크기 (linear complexity)
- 단, 타입(unify의 인자) 크기가 기하급수로 커질 수 있음:

  예)

  let f1 = fn x (x,x) : $\forall a.a \to a \times a$

  f2 = fn x (f1(f1 x)) : $\forall a.a \to (a \times a) \times (a \times a)$

  ...

  - 사람 프로래밍에는 출현하지않는 패턴

메모리주소가 값인 언어

$$
\begin{array}{rrcll}
\text{식} & E & \to & \vdots & \\
 & & | & \texttt{malloc}\, E & \\
 & & | & \texttt{!}\, E & \\
 & & | & E \texttt{ := } E & \\
 & & & & \\
\text{타입} & \tau & \to & \iota & \text{기본타입} \\
 & & | & \alpha & \text{타입변수} \\
 & & | & \tau \to \tau & \\
 & & | & \tau\ loc & \\
 & & & & \\
\text{타입틀} & \sigma & \to & \tau & \text{단순한타입} \\
 & & | & \forall \alpha.\sigma & \text{여러모양타입 (prenex form)}
\end{array}
$$

let-여러모양타입 규칙: 자연스런 확장

$$\frac{\Gamma \vdash E : \tau}{\Gamma \vdash \texttt{malloc}\ E : \tau\ loc}$$

$$\frac{\Gamma \vdash E : \tau\ loc}{\Gamma \vdash\ !E : \tau}$$

$$\frac{\Gamma \vdash E_1 : \tau\ loc \quad \Gamma \vdash E_2 : \tau}{\Gamma \vdash E_1 := E_2 : \tau}$$

그리고 예전그대로

$$\cdots$$

$$\frac{\Gamma \vdash E_1 : \tau \quad \Gamma + x : GEN_\Gamma(\tau) \vdash E_2 : \tau'}{\Gamma \vdash \texttt{let}\ x\ E_1\ E_2 : \tau'}$$

그러나, 안전하지 않은

let

f = malloc (λx.x)

in

f := λx.x+1;

(!f) true

- 실행중 타입 에러.
- 그러나 우리의 타입 시스템은 타입이 있는 것으로 허용.

`let` $x\ E_1\ E_2$

- E_1 실행중에 메모리 주소를 새롭게 할당받는 일이 없는 경우에만 안전
- 식 E_1이 실행중에 메모리를 할당받을지 예측해야
- 실행전 정확히는 예측불가능, 안전하게는 가능

$$
\begin{aligned}
expansive(n) &= false \\
expansive(x) &= false \\
expansive(\texttt{fn}\ x\ E) &= false \\
expansive(E_1\ E_2) &= true \\
expansive(E_1\ \texttt{+}\ E_2) &= expansive(E_1) \vee expansive(E_2) \\
expansive(\texttt{let}\ x\ E_1\ E_2) &= expansive(E_1) \vee expansive(E_2)
\end{aligned}
$$

(너무 안전/보수적?)

let-여러모양타입 규칙: 메모리주소가 값일때

$$\frac{\Gamma \vdash E : \tau}{\Gamma \vdash \texttt{malloc}\ E : \tau\ loc} \qquad \frac{\Gamma \vdash E : \tau\ loc}{\Gamma \vdash\ !E : \tau} \qquad \frac{\Gamma \vdash E_1 : \tau\ loc \quad \Gamma \vdash E_2 : \tau}{\Gamma \vdash E_1\ \texttt{:=}\ E_2 : \tau}$$

$$\frac{\Gamma \vdash E_1 : \tau \quad \Gamma + x : GEN_\Gamma(\tau) \vdash E_2 : \tau'}{\Gamma \vdash \texttt{let}\ x\ E_1\ E_2 : \tau'}\ \neg expansive(E)$$

$$\frac{\Gamma \vdash E_1 : \tau \quad \Gamma + x : \tau \vdash E_2 : \tau'}{\Gamma \vdash \texttt{let}\ x\ E_1\ E_2 : \tau'}\ expansive(E)$$

$$\begin{aligned}
expansive(n \mid x \mid \texttt{fn}\ x\ E) &= false \\
expansive(E_1\ E_2) &= true \\
expansive(E_1\ \texttt{+}\ E_2) &= expansive(E_1) \vee expansive(E_2) \\
expansive(\texttt{let}\ x\ E_1\ E_2) &= expansive(E_1) \vee expansive(E_2)
\end{aligned}$$