오늘 할 것

- 숙제 5 풀이
- 숙제 6 설명

HW5

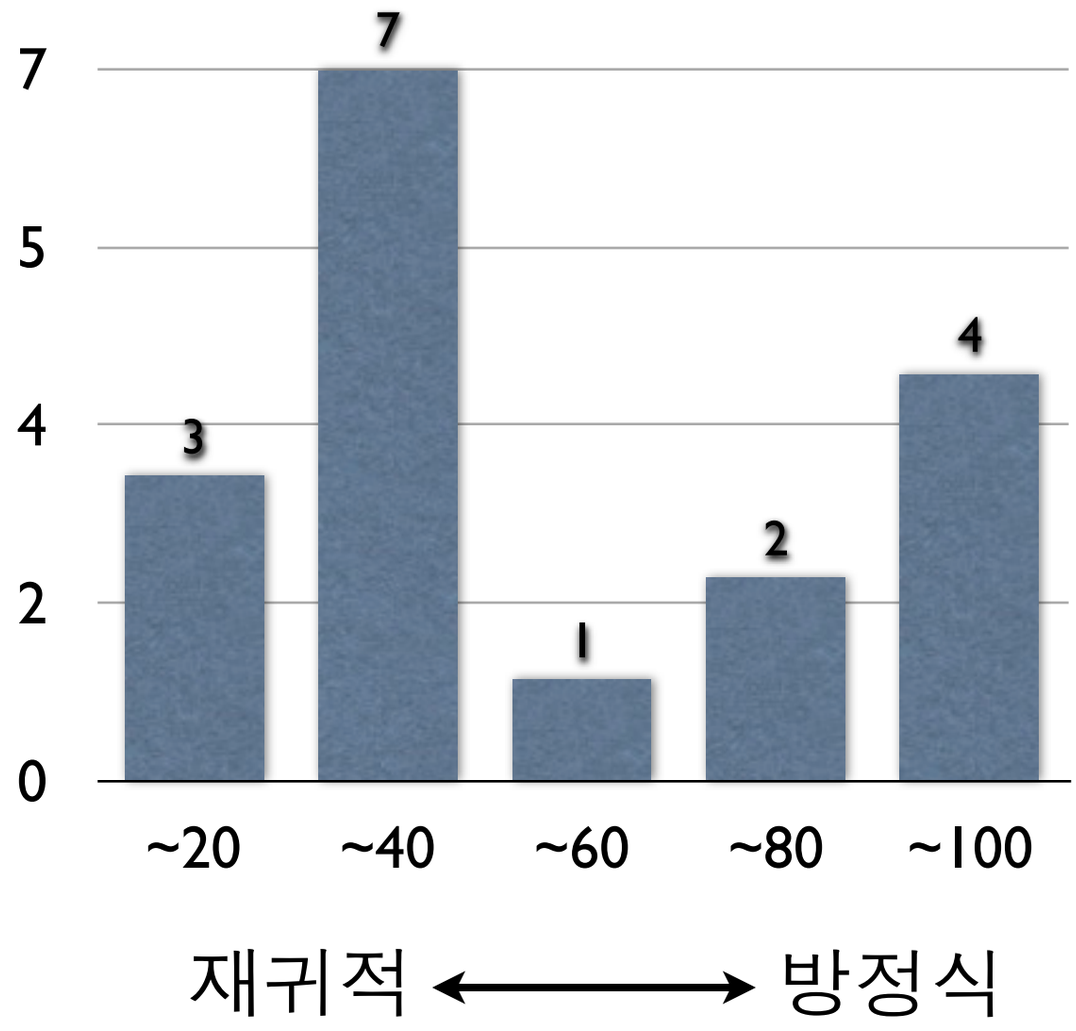
Exercise 1 (60pts) M interpreter with a simple type system

- 수업시간에 구축해간 프로그래밍 언어에 약간의 간을 한 언어가 M 이다. M 언어의 실행기의 대부분이 제공될 것이다. 첨부된 M의 정의에 그 언어의 의미에 대한 모든 것이 있다. 이에 기초해서 나머지를 완성하라.

- 그리고, 주어진 M 실행기에 안전한 단순 타입 시스템(simple type system)을 설치하라. 첨부된 M의 정의에서 타입 시스템의 미흡한 부분을 완성하고, 그것의 충실한 타입 유추기를 만들어서 M 실행기 앞단에 장착하도록 한다. 그러한 M 실행기는 항상 잘 도는 건강한 프로그램만 실행하게 된다. 그래서 건강해진 밥상, M 실행기를 즐길 수 있기를.

Review: 5-2

- 17명 제출
- 어려움
  - 방정식 세우기
  - OR 처리



M 언어

e	::=	*const*	constant
	|	*id*	identifier
	|	`fn` *id* `=>` e	function
	|	e e	application
	|	`let` *bind* `in` e `end`	local block
	|	`if` e `then` e `else` e	branch
	|	e *op* e	infix binary operation
	|	`read`	input
	|	`write` e	output
	|	`(`e`)`	
	|	`malloc` e	allocation
	|	e `:=` e	assignment
	|	`!`e	bang, dereference
	|	e `;` e	sequence
	|	`(`e`,`e`)`	pair
	|	e`.1`	first component
	|	e`.2`	second component

bind	::=	`val` *id* `=` e
	|	`rec` *id* `=` `fn` *id* `=>` e
op	::=	`+` | `-` | `=` | `and` | `or`
const	::=	`true` | `false` | *string* | *num*
id		
string		
num		

M

```
(* syntax of M *)
type exp = CONST of const
         | VAR of id
         | FN of id * exp
         | APP of exp * exp
         | LET of decl * exp
         | IF of exp * exp * exp
         | BOP of bop * exp * exp
         | READ
         | WRITE of exp
         | MALLOC of exp           (*   malloc e *)
         | ASSIGN of exp * exp     (*   e := e   *)
         | BANG of exp             (*   !e       *)
         | SEQ of exp * exp        (*   e ; e    *)
         | PAIR of exp * exp       (*   (e, e)   *)
         | SEL1 of exp             (*   e.1      *)
         | SEL2 of exp             (*   e.2      *)
and const = S of string | N of int | B of bool
and id = string
and decl = REC of id * exp         (*   recursive function    *)
         | NREC of id * exp        (*   non-recursive function  *)
and bop = ADD | SUB | EQ | AND | OR
```

Dynamic Semantics

[Const] $$\sigma, M \vdash const \Rightarrow const \text{ in } Val, M$$

[Id] $$\frac{\sigma(x) = v}{\sigma, M \vdash x \Rightarrow v, M}$$

[Fun] $$\sigma, M \vdash \texttt{fn}\ x\ \texttt{=>}\ e \Rightarrow \langle \lambda x.e, \sigma \rangle, M$$

[App] $$\frac{\sigma, M \vdash e_1 \Rightarrow \langle \lambda x.e, \sigma' \rangle, M' \qquad \sigma, M' \vdash e_2 \Rightarrow v_2, M'' \qquad \sigma'[x \mapsto v_2], M'' \vdash e \Rightarrow v, M'''}{\sigma, M \vdash e_1\ e_2 \Rightarrow v, M'''}$$

[RecApp] $$\frac{\sigma, M \vdash e_1 \Rightarrow \langle f \lambda x.e, \sigma' \rangle, M' \qquad \sigma, M' \vdash e_2 \Rightarrow v_2, M'' \qquad \sigma'[x \mapsto v_2][f \mapsto \langle f \lambda x.e, \sigma' \rangle], M'' \vdash e \Rightarrow v, M'''}{\sigma, M \vdash e_1\ e_2 \Rightarrow v, M'''}$$

[Let] $$\frac{\sigma, M \vdash e_1 \Rightarrow v_1, M' \qquad \sigma[x \mapsto v_1], M' \vdash e_2 \Rightarrow v, M''}{\sigma, M \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2\ \texttt{end} \Rightarrow v, M''}$$

[RecLet] $$\frac{\sigma, M \vdash e_1 \Rightarrow \langle \lambda x.e, \sigma' \rangle, M' \qquad \sigma[f \mapsto \langle f \lambda x.e, \sigma' \rangle], M' \vdash e_2 \Rightarrow v, M''}{\sigma, M \vdash \texttt{let rec}\ f\ \texttt{=}\ e_1\ \texttt{in}\ e_2\ \texttt{end} \Rightarrow v, M''}$$

•••

Static Semantics

Type

$$
\begin{array}{lcll}
\tau & ::= & i & \text{integer type} \\
& | & b & \text{boolean type} \\
& | & s & \text{string type} \\
& | & \tau \times \tau & \text{pair type} \\
& | & \tau \text{ loc} & \text{location type} \\
& | & \tau \rightarrow \tau & \text{function type}
\end{array}
$$

[Num] $\Gamma \vdash n : i$

[Bool] $\Gamma \vdash \texttt{true} : b \qquad \Gamma \vdash \texttt{false} : b$

[String] $\Gamma \vdash \mathit{string} : s$

[Fun] $$\frac{\Gamma[x \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \texttt{fn}\ x\ \texttt{=>}\ e : \tau_1 \rightarrow \tau_2}$$

[App] $$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}$$

[Let] $$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma[x \mapsto \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2\ \texttt{end} : \tau_2}$$

...

추론 알고리즘 구현

타입시스템

- 설계
  - 간단한 단순 타입 시스템 정의
  - 안전성 증명
- 구현
  - 주어진 E가 타입유추되는가?
  - E의 증명을 자동으로 만들기
    - 타입유추기, 증명기

$$\frac{}{\Gamma \vdash n : \iota}$$

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

$$\frac{\Gamma + x : \tau_1 \vdash E : \tau_2}{\Gamma \vdash \lambda x.E : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash E_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash E_2 : \tau_1}{\Gamma \vdash E_1\ E_2 : \tau_2}$$

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

타입체커 구현?

- 타입추론규칙(type inference rule) : 증명규칙
  - 규칙들은 $\vdash E : \tau$ 를 증명하는데 이용
  - 증명되면 E 는 문제없이 도는 프로그램이고 최종결과가 τ 타입
- 그런데, 증명은 누가 하나요?
  - 사람이 손으로?
  - 기계가 자동으로? 타입체커

구현이 간단할까?
(재귀적으로 가능할까?)

$$\frac{}{\Gamma \vdash n : \iota}$$

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

$$\frac{\Gamma \vdash E_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash E_2 : \tau_1}{\Gamma \vdash E_1\ E_2 : \tau_2}$$

$$\frac{\Gamma + x : \tau_1 \vdash E : \tau_2}{\Gamma \vdash \lambda x.E : \tau_1 \to \tau_2}$$

재귀적으로
불가능

```
type id = string
type exp = Num of int
         | Var of id
         | App of exp * exp
         | Abs of id * exp
type typ = Int | Arrow of typ * typ
type tenv = id -> typ
exception Type_error

let rec typeck : tenv -> exp -> typ
=fun te e -> match e with
      Num n -> Int
     |Var x -> te x
     |App (e1,e2) ->
         let Arrow (t1,t2) = typeck te e1 in
         let t1' = typeck te e2
         in  if t1 = t1' then t2
             else raise Type_error
|Abs (id,e) -> ???
```

타입을 명시하면 가능

$$\frac{}{\Gamma \vdash n : \iota}$$

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

$$\frac{\Gamma \vdash E_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash E_2 : \tau_1}{\Gamma \vdash E_1\ E_2 : \tau_2}$$

$$\frac{\Gamma + x : \tau_1 \vdash E : \tau_2}{\Gamma \vdash \lambda x : \tau_1 . E : \tau_1 \to \tau_2}$$

```
type id = string
type typ = Int | Arrow of typ * typ
type exp = Num of int
         | Var of id
         | App of exp * exp
         | Abs of id * typ * exp
type tenv = id -> typ
exception Type_error

let rec typck : tenv -> exp -> typ
=fun te e -> match e with
     Num n -> Int
    |Var x -> te x
    |App (e1,e2) ->
        let Arrow (t1,t2) = typck te e1 in
        let t1' = typck te e2
        in  if t1 = t1' then t2
            else raise Type_error
    |Abs (x,t1,e) ->
        let t2 = typck (te_extend x t te) e
        in  Arrow (t1,t2)
```

타입 유추 알고리즘

- 타입을 자동으로 유추
  - 프로그램에 타입을 명시해 줄 필요가 없는
  - 연립방정식을 도출하고 풀기
    - 프로그램 E의 부품식이 어떤 타입이 되야 할 지에 대한 방정식
    - 풀기: 동일화(unification) 알고리즘

방정식 도출

$$\underbrace{(\lambda x.\underbrace{\underbrace{x}_{1}}_{2})\underbrace{1}_{3}}_{4}$$

$$\begin{aligned}
\alpha_1 &= \alpha_x \\
\alpha_2 &= \alpha \to \beta \\
\alpha_3 &= \iota \\
\alpha &= \alpha_3 \\
\beta &= \alpha_x \\
\alpha_x &= \alpha \\
\alpha_4 &= \beta
\end{aligned}$$

연립방정식 세우기

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

$$
\begin{array}{rcll}
V(\Gamma, n, \tau) & = & \tau \doteq \iota & \\
V(\Gamma, x, \tau) & = & \tau \doteq \tau' & \text{if } x : \tau' \in \Gamma \\
V(\Gamma, E_1 \ \texttt{+} \ E_2, \tau) & = & \tau \doteq \iota \ \wedge \ V(\Gamma, E_1, \iota) \ \wedge \ V(\Gamma, E_2, \iota) & \\
V(\Gamma, \lambda x.E, \tau) & = & \tau \doteq \alpha_1 \rightarrow \alpha_2 \ \wedge \ V(\Gamma + x : \alpha_1, e, \alpha_2) & \text{new } \alpha_1, \alpha_2 \\
V(\Gamma, E_1 \ E_2, \tau) & = & V(\Gamma, E_1, \alpha \rightarrow \tau) \ \wedge \ V(\Gamma, E_2, \alpha) & \text{new } \alpha
\end{array}
$$

연립방정식 세우기 예

$$
\begin{aligned}
& V(\emptyset, (\lambda x.x)\ 1, \tau) \\
= {} & V(\emptyset, \lambda x.x, \alpha \to \tau) \wedge V(\emptyset, 1, \alpha) && \cdots \text{new } \alpha \\
= {} & \alpha \to \tau \doteq \alpha_1 \to \alpha_2 \wedge V(x : \alpha_1, x, \alpha_2) \wedge \alpha \doteq \iota && \cdots \text{new } \alpha_1, \alpha_2 \\
= {} & \alpha \to \tau \doteq \alpha_1 \to \alpha_2 \wedge \alpha_1 \doteq \alpha_2 \wedge \alpha \doteq \iota
\end{aligned}
$$

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

동일화 알고리즘

$$\texttt{unify}(\tau, \tau') : TyVar \overset{\text{fin}}{\hookrightarrow} Type$$

$$\begin{aligned}
\texttt{unify}(\iota, \iota) &= \emptyset \\
\texttt{unify}(\alpha, \tau) \; or \; \texttt{unifty}(\tau, \alpha) &= \{\alpha \mapsto \tau\} \quad \text{if } \alpha \text{ does not occur in } \tau \\
\texttt{unify}(\tau_1 \to \tau_2, \tau_1' \to \tau_2') &= \text{let } S = \texttt{unify}(\tau_1, \tau_1') \\
&\qquad\qquad S' = \texttt{unify}(S\tau_2, S\tau_2') \\
&\qquad \text{in } S'S \\
\texttt{unify}(_) &= \text{fail}
\end{aligned}$$

타입

```
type ty = TVar of int
        | TInt | TString | TBool
        | TPair of ty * ty
        | TLoc of ty
        | TArrow of ty * ty

let next_tvar = ref 0
let newt() = next_tvar := !next_tvar + 1; TVar !next_tvar
```

치환

타입변수를 타입으로 바꾸는 '치환'들의 집합

ex) $S = \{\alpha_1 \hookrightarrow int, \alpha_2 \hookrightarrow int\}$

치환의 적용
: 타입안의 (자유)타입변수를 바꿔치기

ex) $\{\alpha_1 \hookrightarrow int, \alpha_2 \hookrightarrow bool\}\ (\alpha_1 \rightarrow \alpha_2) = int \rightarrow int$

치환의 타입: type→type

치환의 생성 / 합성

타입변수(x)를 타입(tau)로 바꿔치기
하는 함수(치환)을 리턴

```
let subst : int -> ty -> (ty -> ty) =
fun x tau ->
    let rec s : ty -> ty
    =fun t ->
        match t with
          TVar y -> if y = x then tau else t
        | TPair (t1, t2) -> TPair (s t1, s t2)
        | TArrow (t1, t2) -> TArrow (s t1, s t2)
        | TLoc t' -> TLoc (s t')
        | TInt | TString | TBool -> t
    in s

let ( @@ ) g f = (fun t -> g (f t))
let id = (fun x -> x)
```

빈 치환

두 치환을 합성
g @@ f

cf) 하나만 바꾸는 치환

우리가 구현한 치환은 한번에 여러 변수를 바꾸는 치환을 만들수 없다

$$\{\alpha \hookrightarrow \beta, \beta \hookrightarrow \alpha\}\ (\alpha) = \beta$$
$$\{\beta \hookrightarrow \alpha\}\{\alpha \hookrightarrow \beta\}\ (\alpha) = \alpha$$

이런경우는 고려하지 않아도 되나?

동일화 알고리즘

$$\texttt{unify}(\tau, \tau') : TyVar \overset{\text{fin}}{\hookrightarrow} Type$$

$$\begin{array}{rcl}
\texttt{unify}(\iota, \iota) & = & \emptyset \\
\texttt{unify}(\alpha, \tau)\ or\ \texttt{unifty}(\tau, \alpha) & = & \{\alpha \mapsto \tau\} \quad \text{if } \alpha \text{ does not occur in } \tau \\
\texttt{unify}(\tau_1 \to \tau_2, \tau_1' \to \tau_2') & = & \text{let } S = \texttt{unify}(\tau_1, \tau_1') \\
 & & \quad S' = \texttt{unify}(S\tau_2, S\tau_2') \\
 & & \text{in } S'S \\
\texttt{unify}(_) & = & \text{fail}
\end{array}$$

항상 하나의 엔트리만 가지도록 생성

엔트리 하나짜리 치환들이 합성됨

타입환경

```
let ( @+ ) g (x, t) = fun y -> if y = x then t else g y
let emptyG = fun x -> raise (TypeError ("Unknown id: " ^ x))
```

```
type ty = TVar of int
        | TInt | TString | TBool
        | TPair of ty * ty
        | TLoc of ty
        | TArrow of ty * ty
```

동일화 알고리즘
(unification algorithm)

```
let rec unify : ty -> ty -> (ty -> ty)
=fun t1 t2 ->
    match (t1, t2) with
      (TVar x, tau) ->
        if TVar x = tau then id
        else if not (occurs x tau) then subst x tau
        else raise TypeError
    | (tau, TVar x) -> unify (TVar x) tau
    | (TPair (a, b), TPair (c, d)) -> unifypair (a, b) (c, d)
    | (TArrow (a, b), TArrow (c, d)) -> unifypair (a, b) (c, d)
    | (TLoc t, TLoc t') -> unify t t'
    | (tau, tau') ->
        if tau = tau' then id
        else raise TypeError
and unifypair (t1, t2) (t1', t2') =
    let s = unify t1 t1' in
    let s' = unify (s t2) (s t2') in
        s' @@ s
```

$$
\begin{aligned}
\texttt{unify}(\iota, \iota) &= \emptyset \\
\texttt{unify}(\alpha, \tau) \; or \; \texttt{unifty}(\tau, \alpha) &= \{\alpha \mapsto \tau\} \quad \text{if } \alpha \text{ does not occur in } \tau \\
\texttt{unify}(\tau_1 \to \tau_2, \tau_1' \to \tau_2') &= \text{let } S = \texttt{unify}(\tau_1, \tau_1') \\
&\qquad\qquad S' = \texttt{unify}(S\tau_2, S\tau_2') \\
&\qquad\qquad \text{in } S'S \\
\texttt{unify}(\_) &= \text{fail}
\end{aligned}
$$

```
let rec unify : ty -> ty -> (ty -> ty)
=fun t1 t2 ->
    match (t1, t2) with
      (TVar x, tau) ->
        if TVar x = tau then id
        else if not (occurs x tau) then subst x tau
        else raise TypeError
    | (tau, TVar x) -> unify (TVar x) tau
    | (TPair (a, b), TPair (c, d)) -> unifypair (a, b) (c, d)
    | (TArrow (a, b), TArrow (c, d)) -> unifypair (a, b) (c, d)
    | (TLoc t, TLoc t') -> unify t t'
    | (tau, tau') ->
        if tau = tau' then id
        else raise TypeError
and unifypair (t1, t2) (t1', t2') =
    let s = unify t1 t1' in
    let s' = unify (s t2) (s t2') in
        s' @@ s
```

타입방정식 확장

$$
\begin{array}{rrcll}
\mathit{TyEqn} & u & \rightarrow & \tau \doteq \tau & \text{타입 방정식} \\
 & & \mid & u \wedge u & \text{연립} \\
\mathit{Type} & \tau & \rightarrow & \alpha \in \mathit{TyVar} & \text{타입 변수} \\
 & & \mid & \iota \mid \tau \rightarrow \tau &
\end{array}
$$

u ∨ u 도 필요?

[Write]
$$\frac{\Gamma \vdash e : \tau \quad \tau = i, b, \text{ or } s}{\Gamma \vdash \texttt{write}\ e : \tau}$$

[Op]
$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau \qquad \tau = i, b, s, \text{ or } l}{\Gamma \vdash e_1 \texttt{ = } e_2 : b}$$

cf) 간단한 경우면 충분

일반적인 형태를 고려할 필요 없다

즉, 이런 방정식은 만들어지지 않음

$$(u \vee u) \wedge ((u \wedge u) \vee u)$$

$\wedge$는 $\vee$ 안에 등장하지 않음을 가정

$$(u \vee u) \wedge (u \vee u) \wedge u$$

$$u \rightarrow \tau = \tau \mid \tau = [\tau_1, \tau_2, \ldots, \tau_n] \mid u \wedge u$$

방정식 도출

방정식

```
type constraints = U of exp * ty * ty
                 | Or of exp * ty * ty list
```

연립방정식은 constraints 의 리스트

```
let rec v : (id -> ty) -> exp -> ty -> constraints list -> constraints list
=fun g exp tau ->
    let u tau tau' = (fun c -> U (exp, tau, tau') :: c) in
    let mkor tau taul = (fun c -> Or (exp, tau, taul) :: c) in
        match exp with
          CONST (S _) -> u tau TString
        | CONST (N _) -> u tau TInt
        | CONST (B _) -> u tau TBool
        | VAR x -> u tau (g x)
        | FN (x, e) ->
            let t1 = newt() in
            let t2 = newt() in
                u tau (TArrow (t1, t2)) @@ v (g @+ (x, t1)) e t2
        | APP (e1, e2) ->
            let t1 = newt() in
                v g e1 (TArrow (t1, tau)) @@ v g e2 t1
```

방정식 도출

```
| BOP (op, e1, e2) ->
    let (t, t') = match op with ADD | SUB -> (TInt, TInt)
                              | AND | OR -> (TBool, TBool)
                              | EQ -> (newt(), TBool)
    in
        u tau t' @@ mkor t [TInt; TString; TBool; TLoc (newt())] @@
        v g e2 t @@ v g e1 t
| READ -> u tau TInt
| WRITE e -> mkor tau [TInt; TString; TBool] @@ v g e tau
```

방정식 풀기

```
let val x = 1
in
    write x
end
```

Or (a1, [string,bool,int]) ^ U (a1, a2) ^ U (a2, int)

의 해는 아래 세 방정식 중 하나를 만족하는 해

U(a1,string) ^ U (a1, a2) ^ U (a2, int)
U(a1,bool) ^ U (a1, a2) ^ U (a2, int)
U(a1,int) ^ U (a1, a2) ^ U (a2, int)

순서 변경

`Or (a1, [string,bool,int]) ^ U (a1, a2) ^ U (a2, int)`

vs.

`U (a1, a2) ^ U (a2, int) ^ Or (a1, [string,bool,int])`

```
(* reorder: constraints list -> constraints list *)
let reorder c =
    let (u, o) = List.partition (function Or _ -> false | _ -> true) c in
                 u @ o
```

방정식 풀기

```
let rec c2s : (ty -> ty) -> constraints list -> (ty -> ty)
=fun s cl ->
    match cl with
     [] -> s
    | (U (e, t1, t2)::c) -> c2s (unify (s t1) (s t2) @@ s) c
    | (Or (e, t1, tl)::c) ->
let rec tt =
    function [] -> raise TypeError
    | (h::t) -> try c2s s (U (e, t1, h)::c)
                with _ -> tt t
in  tt tl
```

최종 타입

```
((c2s id (reorder (v emptyG exp tau))) tau)
```

HW6

Exercise 1 (50pts) "저지방 고단백 M"[1]

M 실행기 위에, 지난 숙제에서 구현한 단순 타입 시스템(simple type system)을 대신해서 보다 정교한 let-다형 타입 시스템(let-polymorphic type system)을 장착하자. 그래서, 단순 타입 시스템이 받아들이는 프로그램은 물론이고, 다형 타입의 함수(polymorphic function, 다양한 타입의 인자에 적용될 수 있는 함수)를 가지는 프로그램까지 받아들여 지도록.

예를들어, 아래와 같은 잘 도는 프로그램들이 단순 타입 시스템에서는 받아들여지지 않았으나, let-다형 타입 시스템에서는 받아들여지게 된다.

TA가 제공하는 M 실행기의 틀 위에 let-다형 타입 시스템을 장착하라.

Due: 6/3(Fri), 24:00

```
(* example 1: polymorphic toys *)

   let val I = fn x => x
       val add = fn x => x.1 + x.1
       val const = fn n => 10
   in
     I I;
     add(1, true) + add(2, "snu 310 fall 2009");
     const 1 + const true + const "kwangkeun yi"
   end
```

```
(* example 2: polymorphism with imperatives *)

    let val f = fn x => malloc x
    in
       let val a = f 10
           val b = f "pl"
           val c = f true
       in
         a := !a + 1;
         b := "hw7";
         c := !c or false
       end
end
```

```
(* example 3: polymorphic swap *)

   let val swap =
           fn order_pair =>
              if (order_pair.1) (order_pair.2)
               then (order_pair.2)
                  else (order_pair.2.2, order_pair.2.1)
in
  swap(fn pair => pair.1 + 1 = pair.2, (1,2));
  swap(fn pair => pair.1 or pair.2, (true, false))
end
```

```
(* S K I combinators *)

   let val I = fn x => x
       val K = fn x => fn y => x
       val S = fn x => fn y => fn z => (x z) (y z)
   in
    S (K (S I)) (S (K K) I) 1 (fn x => x+1)
   end
```