COSE212: Programming Languages

Lecture 13 — Automatic Type Inference (1)

Hakjoo Oh 2020 Fall

The Problem of Automatic Type Inference

Given a program E, infer the most general type of E if E can be typed (i.e., $[] \vdash E : t$ for some $t \in T$). If E cannot be typed, say so.

- let $f = \operatorname{proc}(x)(x+1)$ in $(\operatorname{proc}(x)(x1)) f$
- let f = proc (x) (x + 1) in (proc (x) (x true)) f
- ullet proc (x) x

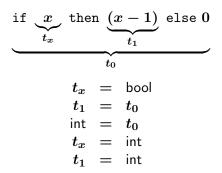
Automatic Type Inference

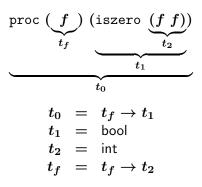
- A static analysis algorithm that automatically figures out types of expressions by observing how they are used.
- The algorithm is *sound and complete* with respect to the type system design.
 - (Sound) If the analysis finds a type for an expression, the expression is well-typed with the type according to the type system.
 - ► (Complete) If an expression has a type according to the type system, the analysis is guaranteed to find the type.
- The algorithm consists of two steps:
 - Generate type equations from the program text.
 - Solve the equations.

Generating Type Equations

For every subexpression and variable, introduce type variables and derive equations between the type variables.

$$t_0 = t_f
ightarrow t_1 \ t_1 \ t_2 \ t_3 \ t_4 \ t_5 = t_1 \ t_5 \ t_1 \ t_5 \ t_1 \ t_5 \ t$$





Idea: Deriving Equations from Typing Rules

For each expression e and variable x, let t_e and t_x denote the type of the expression and variable. Then, the typing rules dictate the equations that must hold between the type variables.

$$egin{array}{cccc} rac{\Gamma dash E_1: ext{int}}{\Gamma dash E_1 + E_2: ext{int}} \ & t_{E_1} = ext{int} \ & t_{E_2} = ext{int} \ & t_{E_1 + E_2} = ext{int} \end{array}$$

$$egin{array}{c} \Gamma dash E : \mathsf{int} \ \Gamma dash \mathsf{iszero} \ E : \mathsf{bool} \end{array}$$

$$t_E = \mathsf{int} \ \land \ t_{(\mathsf{iszero}\ E)} = \mathsf{bool}$$

$$egin{aligned} \Gamma dash E_1: t_1
ightarrow t_2 & \Gamma dash E_2: t_1 \ & \Gamma dash E_1 E_2: t_2 \ & t_{E_1} = t_{E_2}
ightarrow \ t_{(E_1 \ E_2)} \end{aligned}$$

Idea: Deriving Equations from Typing Rules

$$\begin{array}{lll} \Gamma \vdash E_1 : \mathsf{bool} & \Gamma \vdash E_2 : t & \Gamma \vdash E_3 : t \\ \hline \Gamma \vdash \mathsf{if} \ E_1 \ \mathsf{then} \ E_2 \ \mathsf{else} \ E_3 : t \\ & t_{E_1} &= \ \mathsf{bool} \ \land \\ & t_{E_2} &= \ t_{(\mathsf{if} \ E_1 \ \mathsf{then} \ E_2 \ \mathsf{else} \ E_3)} \ \land \\ & t_{E_3} &= \ t_{(\mathsf{if} \ E_1 \ \mathsf{then} \ E_2 \ \mathsf{else} \ E_3)} \end{array} \land \\ & \underbrace{ \begin{bmatrix} x \mapsto t_1 \end{bmatrix} \Gamma \vdash E : t_2 }_{\Gamma \vdash \mathsf{proc} \ x \ E : t_1 \to t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{proc} \ x \ E : t_1 \to t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{proc} \ x \ E : t_1 \to t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{proc} \ x \ E : t_1 \to t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{in} \ E_2 : t_2} \\ & \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{let} \ x = E_1 \ \mathsf{let} \ \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{let} \ \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{let} \ \underbrace{ \begin{matrix} (x \mapsto t_1) \Gamma \vdash E_2 : t_2 \end{matrix} }_{\Gamma \vdash \mathsf{let} \ x = E_1 \ \mathsf{let} \$$

Summary

The algorithm for automatic type inference:

- Generate type equations from the program text.
 - Introduce type variables for each subexpression and variable.
 - ► Generate equations between type variables according to typing rules.
- Solve the equations.

COSE212: Programming Languages

Lecture 14 — Automatic Type Inference (2)

Hakjoo Oh 2020 Fall

Goal

- So far we have informally discussed how to derive type equations.
- In this lecture, we define the procedure precisely.

Language

Type Equations

Type equations are conjunctions of "type equalities": e.g.,

$$egin{array}{lll} t_0 &=& t_f
ightarrow t_1 \ t_1 &=& t_x
ightarrow t_4 \ t_3 &=& {
m int} \ t_4 &=& {
m int} \ t_2 &=& {
m int} \ t_f &=& {
m int}
ightarrow t_3 \ t_f &=& t_x
ightarrow t_4 \ \end{array}$$

• Type equations (TyEqn) are defined inductively:

$$\begin{array}{ccc} \mathit{TyEqn} & \to & \emptyset \\ & | & \mathit{T} \doteq \mathit{T} \ \land \ \mathit{TyEqn} \end{array}$$

Deriving Type Equations

Algorithm for generating equations:

$$\mathcal{V}: (\mathit{Var} o T) imes E imes T o \mathit{TyEqn}$$

• $\mathcal{V}(\Gamma,e,t)$ generates the condition for e to have type t in Γ :

$$\Gamma \vdash e:t$$
 iff $\mathcal{V}(\Gamma,e,t)$ is satisfied.

- Examples:
 - $\mathcal{V}([x \mapsto \text{int}], x+1, \alpha) =$
 - $ightharpoonup \mathcal{V}(\emptyset, \operatorname{proc}(x) \ (\operatorname{if} x \ \operatorname{then} \ 1 \ \operatorname{else} \ 2), lpha
 ightarrow eta) =$
- ullet To derive type equations for closed expression E, we call $\mathcal{V}(\emptyset, E, \alpha)$, where α is a fresh type variable.

Deriving Type Equations

```
 \begin{array}{l} \mathcal{V}(\emptyset, (\operatorname{proc}\;(x)\;(x))\; 1, \alpha) \\ = \mathcal{V}(\emptyset, \operatorname{proc}\;(x)\;(x), \alpha_1 \to \alpha) \wedge \mathcal{V}(\emptyset, 1, \alpha_1) & \operatorname{new}\; \alpha_1 \\ = \alpha_1 \to \alpha \doteq \alpha_2 \to \alpha_3 \wedge \mathcal{V}([x \mapsto \alpha_2], x, \alpha_3) \wedge \alpha_1 \doteq \operatorname{int} & \operatorname{new}\; \alpha_2, \alpha_3 \\ = \alpha_1 \to \alpha \doteq \alpha_2 \to \alpha_3 \wedge \alpha_2 \doteq \alpha_3 \wedge \alpha_1 \doteq \operatorname{int} \end{array}
```

Exercise 1

$$\mathcal{V}(\emptyset, \mathtt{proc}\; (f)\; (f\; 11), lpha)$$

Exercise 2

$$\mathcal{V}([x\mapsto \mathsf{bool}], \mathsf{if}\ x\ \mathsf{then}\ (x-1)\ \mathsf{else}\ 0, lpha)$$

Exercise 3

$$\mathcal{V}(\emptyset, \mathtt{proc}\; (f)\; (\mathtt{iszero}\; (f\; f)), lpha)$$

Summary

We have defined the algorithm for deriving type equations from program text:

- ullet Given a program E, call $\mathcal{V}(\emptyset,E,lpha)$ to derive type equations.
- ullet Solve the equations and find the type assigned to lpha.

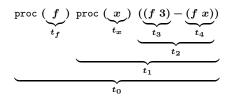
COSE212: Programming Languages

Lecture 15 — Automatic Type Inference (3)

Hakjoo Oh 2020 Fall

Finding a Solution of Type Equations

Find the values of type variables that make all the equations true.



Equations			Solution			
t_0	=	$t_f ightarrow t_1$	t_0	=	$(int \to int) \to (int \to int)$	
t_1	=	$t_x ightarrow t_2$	t_1	=	int o int	
t_3	=	int	t_2	=	int	
t_4	=	int	t_3	=	int	
t_2	=	int	t_4	=	int	
t_f	=	$int \to t_3$	t_f	=	int o int	
t_f	=	$t_x ightarrow t_4$	$ t_x $	=	int	

Static type systems find such a solution using unification algorithm.

The calculation is split into equations to be solved and substitution found so far. Initially, the substitution is empty:

Equations	Substitution
$t_0 = t_f ightarrow t_1$	
$t_1 \; = \; t_x ightarrow t_2$	
$t_3 = int$	
$t_4 = int$	
t_{2} = int	
t_f $=$ int $ o t_3$	
$t_f = t_x ightarrow t_4$	

Consider each equation in turn. If the equation's left-hand side is a variable, move it to the substitution:

		Equations	Substitution		
t_1	=	$t_x ightarrow t_2$	t_0	=	$t_f ightarrow t_1$
t_3	=	int			
t_4	=	int			
$\boldsymbol{t_2}$	=	int			
t_f	=	$int \to t_3$			
t_f	=	$t_x ightarrow t_4$			

Move the next equation to the substitution and propagate the information to the existing substitution (i.e., substitute the right-hand side for each occurrence of t_1):

	Equations	Substitution			
t_3 =		t_0	=	$t_f ightarrow (t_x ightarrow t_2)$	
t_4 =	= int	t_1	=	$t_x ightarrow t_2$	
t_2 =	= int				
t_f =	= int $ ightarrow$ t_3				
t_f =	$t_x ightarrow t_4$				

Same for the next three equations:

Equations	Substitution
$t_4 \; = \; int$	$\mid t_0 \mid = \mid t_f \rightarrow (t_x \rightarrow t_2)$
t_{2} = int	$egin{array}{cccc} t_1 &=& t_x ightarrow t_2 \end{array}$
t_f $=$ int $ o t_3$	t_3 = int
$t_f^{'} = t_x ightarrow t_4$	
Equations	Substitution
t_2 = int	$t_0 = t_f ightarrow (t_x ightarrow t_2)$
t_f $=$ int $ o t_3$	$t_1 = t_x ightarrow t_2$
$t_f^{-}=t_x ightarrow t_4$	t_3 = int
·	$ t_4 = {int}$
Equations	Substitution
$t_f = {\sf int} o t_3$	$t_0 = t_f ightarrow (t_x ightarrow { m int})$
$t_f = t_x ightarrow t_4$	$t_1 = t_x ightarrow int$
-	$\mid t_3 \mid = \mid$ int
	$ig t_4 = int$
	$t_2 = int$

Consider the next equation $t_f={\rm int}\to t_3$. The equation contains t_3 , which is already bound to int in the substitution. Substitute int for t_3 in the equation. This is called *applying* the substitution to the equation.

Equations	Substitution			
$t_f = int o int$	$t_0 = t_f ightarrow (t_x ightarrow int)$			
$t_f = t_x ightarrow t_4$	$egin{array}{cccc} t_1 &=& t_x ightarrow { m int} \end{array}$			
	$t_3 = int$			
	$t_4 = int$			
	$egin{array}{lll} t_0 &=& t_f ightarrow (t_x ightarrow { m int}) \ t_1 &=& t_x ightarrow { m int} \ t_3 &=& { m int} \ t_4 &=& { m int} \ t_2 &=& { m int} \ \end{array}$			

Move the resulting equation to the substitution and update it.

	Equations		Substitution				
$t_f =$	$t_x ightarrow t_4$	t_0	=	$(int o int) o (t_x o int)$			
		$ t_1 $	=	$t_{m{x}} ightarrow int$			
		t_3	=	int			
		t_4	=	int			
		t_2	=	int			
		$\mid t_f \mid$	=	$(\operatorname{int} o \operatorname{int}) o (t_x o \operatorname{int})$ $t_x o \operatorname{int}$ int int int int int $\operatorname{int} o \operatorname{int}$			

Apply the substitution to the equation:

Equations	Substitution			
$int o int \ = \ t_x o int$	$t_0 = (int o int) o (t_x o int)$			
	$t_1 = t_x o int$			
	$t_3 = int$			
	t_4 = int			
	t_2 = int			
	$egin{array}{lll} t_0 &=& (\operatorname{int} ightarrow \operatorname{int}) ightarrow (t_x ightarrow \operatorname{int}) \ t_1 &=& t_x ightarrow \operatorname{int} \ t_3 &=& \operatorname{int} \ t_4 &=& \operatorname{int} \ t_2 &=& \operatorname{int} \ t_f &=& \operatorname{int} ightarrow \operatorname{int} i$			

If neither side of the equation is a variable, simplify the equation by yielding two new equations:

Equations					Substitution
int	=	t_x	t_0	=	$(int o int) o (t_x o int)$
int	=	int	t_1	=	$t_{m{x}} ightarrow int$
			t_3	=	int
			t_4	=	int
			t_2	=	int
			t_f	=	$(\operatorname{int} o \operatorname{int}) o (t_x o \operatorname{int})$ $t_x o \operatorname{int}$ int int int int int $\operatorname{int} o \operatorname{int}$

Switch the sides of the first equation and move it to the substitution:

Equations	Substitution			
int = int	t_0	=	$(int \to int) \to (int \to int)$	
	t_1	=	$int \to int$	
	t_3	=	int	
	t_4	=	int	
	t_2	=	int	
	t_f	=	$int \to int$	
	$t_{m{x}}$	=	$\begin{array}{l} \operatorname{int} \to \operatorname{int} \\ \operatorname{int} \end{array}$	

The final substitution is the solution of the original equations.

$$t_0 = t_f
ightarrow t_1 \ t_0 = t_f
ightarrow t_1 \ t_f = \operatorname{int}
ightarrow t_1$$

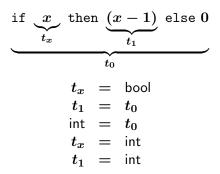
2

Equations Substitution $t_0 = t_f
ightarrow t_1 \ t_f = \operatorname{int}
ightarrow t_1$

Equations	Substitution		
$t_f = int o t_1$	$t_0 = t_f ightarrow t_1$		

Equations Substitution
$$\begin{array}{ccc} t_0 & = & (\mathsf{int} \to t_1) \to t_1 \\ t_f & = & \mathsf{int} \to t_1 \end{array}$$

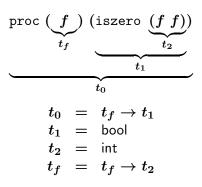
The type is *polymorphic* in t_1 .



The equations have no solutions because, during the unification algorithm, we encounter the following contradictory state:

	Substitution					
bool	=	int		t_x	=	bool
t_1	=	int		t_1	=	int
				t_0	=	int

Because bool and int cannot be equal, there is no solution to the equations.



Example 4

Solving as usual, we encounter a problem:

Equations	Substitution
$t_f = t_f o ext{int}$	$egin{array}{lll} t_0 &=& t_f ightarrow {\sf bool} \ t_1 &=& {\sf bool} \ t_2 &=& {\sf int} \end{array}$

- There is no type t_f that satisfies the equation, because the right-hand side of the equation is always larger than the left.
- If we ever deduce an equation of the form $t=\dots t\dots$ where the type variable t occurs in the right-hand side, we must conclude that there is no solution. This is called *occurrence check*.

Unification Algorithm

For each equation in turn,

- Apply the current substitution to the equation.
- If the equation is always true (e.g. int = int), discard it.
- If the left- and right-hand sides are contradictory (e.g. bool = int), the algorithm fails.
- If neither side is a variable (e.g. int $\rightarrow t_1 = t_2 \rightarrow$ bool), simplify the equation, which eventually generates an equation whose left- or right-hand side is a variable.
- If the left-hand side is not a variable, switch the sides.
- If the left-hand side variable occurs in the right-hand side, the algorithm fails.
- Otherwise, move it to the substitution and substitute the right-hand side for each occurrence of the variable in the substitution.

let
$$x = 4$$
 in $(x \ 3)$

$$let f = proc (z) z in proc (x) ((f x) - 1)$$

let p = iszero 1 in if p then 88 else 99

let f = proc (x) x in if (f (iszero0)) then (f 11) else (f 22)

Substitution

Solutions of type equations are represented by substitution:

$$S \in Subst = \mathit{TyVar} o T$$

Applying a substitution to a type:

$$S(\mathsf{int}) = \mathsf{int}$$
 $S(\mathsf{bool}) = \mathsf{bool}$
 $S(lpha) = egin{cases} t & \mathsf{if} \ lpha \mapsto t \in S \ lpha & \mathsf{otherwise} \end{cases}$
 $S(T_1 o T_2) = S(T_1) o S(T_2)$

Example

Applying the substitution

$$S = \{t_1 \mapsto \mathsf{int}, t_2 \mapsto \mathsf{int} \to \mathsf{int}\}$$
 to to the type $(t_1 \to t_2) \to (t_3 \to \mathsf{int})$: $S((t_1 \to t_2) \to (t_3 \to \mathsf{int}))$ $= S(t_1 \to t_2) \to S(t_3 \to \mathsf{int})$ $= (S(t_1) \to S(t_2)) \to (S(t_3) \to S(\mathsf{int}))$ $= (\mathsf{int} \to (\mathsf{int} \to \mathsf{int})) \to (t_3 \to \mathsf{int})$

Unification

Update the current substitution with equality $t_1 \doteq t_2$.

$$\mathsf{unify}: T \times T \times Subst \to Subst$$

$$\begin{array}{rcl} & \mathsf{unify}(\mathsf{int},\mathsf{int},S) & = & S \\ & \mathsf{unify}(\mathsf{bool},\mathsf{bool},S) & = & S \\ & \mathsf{unify}(\alpha,\alpha,S) & = & S \\ & \mathsf{unify}(\alpha,t,S) & = & \left\{ \begin{array}{l} \mathsf{fail} & \alpha \; \mathsf{occurs} \; \mathsf{in} \; t \\ \mathsf{extend} \; S \; \mathsf{with} \; \alpha \doteq t \; \mathsf{otherwise} \end{array} \right. \\ & \mathsf{unify}(t,\alpha,S) & = \; \mathsf{unify}(\alpha,t,S) \\ & \mathsf{unify}(t_1 \to t_2,t_1' \to t_2',S) & = \; \mathsf{let} \; S' = \mathsf{unify}(t_1,t_1',S) \; \mathsf{in} \\ & \mathsf{let} \; S'' = \mathsf{unify}(S'(t_2),S'(t_2'),S') \; \mathsf{in} \\ & S'' \\ & \mathsf{unify}(\cdot,\cdot,\cdot) & = \; \mathsf{fail} \end{array}$$

- $\operatorname{unify}(\alpha, \operatorname{int} \to \operatorname{int}, \emptyset) =$
- unify(α , int $\rightarrow \alpha$, \emptyset) =
- unify($\alpha \to \beta$, int \to int, \emptyset) =
- $\operatorname{unify}(\alpha \to \beta, \operatorname{int} \to \alpha, \emptyset) =$

Solving Equations

$$\begin{array}{rcl} \text{unifyall}: TyEqn \rightarrow Subst \rightarrow Subst \\ & \text{unifyall}(\emptyset,S) &=& S \\ \text{unifyall}((t_1 \doteq t_2) \ \land \ u,S) &=& \text{let } S' = \text{unify}(S(t_1),S(t_2),S) \\ & \text{in unifyall}(u,S') \end{array}$$

Let \mathcal{U} be the final unification algorithm:

$$\mathcal{U}(u) = \mathsf{unifyall}(u,\emptyset)$$

$\mathsf{typeof}: E \to T$

The final type inference algorithm that composes equation derivation (\mathcal{V}) and equation solving (\mathcal{U}) :

$$\begin{array}{l} \operatorname{typeof}(E) = \\ \operatorname{let} S = \mathcal{U}(\mathcal{V}(\emptyset, E, \alpha)) \quad (\text{new } \alpha) \\ \operatorname{in} S(\alpha) \end{array}$$

Examples

- typeof((proc(x) x) 1)
- typeof(let x = 1 in proc(y) (x + y))

Summary

Automatic type inference:

- derive type equations from the program text, and
- solve the equations by unification algorithm.