

IS893: Advanced Software Security

10. Type Systems

Kihong Heo



Type Systems

- A specialized framework: static analysis by proof construction
- Most widely used form of static analysis
 - Part of many modern languages (e.g., OCaml, Rust, Java, etc)
 - Advancing existing languages (e.g., TypeScript, Hack)
- Assumption: proof construction in a finite proof system
 - Finite proof system = a finite set of inference rules for a predefined set of judgements (i.e., abstract domains are finite sets)

Judgement

- A simple language:

$$E \rightarrow n \mid x \mid \lambda x.E \mid E E \mid E + E$$

- Simple types:

$$\tau \rightarrow int \mid \tau \rightarrow \tau$$

- Judgement: “Expression E has type τ under set Γ of type assumptions”

$$\Gamma \vdash E : \tau$$

Type assumptions for
the free variables in E

Type Inference

$$\frac{}{\Gamma \vdash n : int} \quad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad \frac{\Gamma + x : \tau_1 \vdash E : \tau_2}{\Gamma \vdash \lambda x.E : \tau_1 \rightarrow \tau_2}$$

$$\frac{\Gamma \vdash E_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash E_2 : \tau_1}{\Gamma \vdash E_1 E_2 : \tau_2} \quad \frac{\Gamma \vdash E_1 : int \quad \Gamma \vdash E_2 : int}{\Gamma \vdash E_1 + E_2 : int}$$

Theorem (Soundness). Let E be a program, an expression without free variables. If $\emptyset \vdash E : \tau$, then the program runs without a type error and returns a value of type τ if terminates.

Example

- Program: $(\lambda x.x + 1)(2)$
- The program is typed int because we can prove $\emptyset \vdash (\lambda x.x + 1)(2) : int$
- Proof:

$$\frac{\frac{\frac{x : int \in \{x : int\}}{\{x : int\} \vdash x : int} \quad \{x : int\} \vdash 1 : int}{\{x : int\} \vdash x + 1 : int}}{\emptyset \vdash \lambda x.x + 1 : int \rightarrow int} \quad \emptyset \vdash 2 : int$$
$$\frac{}{\emptyset \vdash (\lambda x.x + 1)(2) : int}$$

Soundness & Expressiveness

- Sound type system:
 - The program is provable \Rightarrow the program satisfies the proven judgement
- Need more precise analysis?
 - Design new proof rules (e.g., polymorphic type systems)

Type Inference

- Type inference = collecting type equations + solving type equations
- Unification algorithm: an efficient algorithm for type inference
 - No iterative computation
- Analogy: a system of equations (constants: types, variables: type variables)

$$\begin{cases} 3x + y = 11 \\ 2x + y = 8 \end{cases} \longrightarrow y = 11 - 3x \longrightarrow 2x + (11 - 3x) = 8 \longrightarrow \begin{matrix} x = 3 \\ y = 2 \end{matrix}$$

Unification (1)

$$\begin{cases} 3x + y = 11 \\ 2x + y = 8 \end{cases}$$

- Collecting: given a program E , $V(\emptyset, E, \alpha)$ returns type equations
- Type variable α : unknown types

$$V(\Gamma, n, \tau) = \{\tau \doteq int\}$$

$$V(\Gamma, x, \tau) = \{\tau \doteq \Gamma(x)\}$$

$$V(\Gamma, \lambda x.E, \tau) = \{\tau \doteq \alpha_1 \rightarrow \alpha_2\} \cup V(\Gamma + x : \alpha_1, E, \alpha_2) \quad (\text{new } \alpha_i)$$

$$V(\Gamma, E_1 E_2, \tau) = V(\Gamma, E_1, \alpha \rightarrow \tau) \cup V(\Gamma, E_2, \alpha) \quad (\text{new } \alpha)$$

$$V(\Gamma, E_1 + E_2, \tau) = \{\tau \doteq int\} \cup V(\Gamma, E_1, int) \cup V(\Gamma, E_2, int)$$

Example

$$\begin{cases} 3x + y = 11 \\ 2x + y = 8 \end{cases}$$

- $\lambda x. x + 1$

$$\begin{aligned} V(\emptyset, \lambda x. x + 1, \alpha) &= \{\alpha \dot{=} \alpha_1 \rightarrow \alpha_2\} \cup V(\{x : \alpha_1\}, x + 1, \alpha_2) \\ &= \{\alpha \dot{=} \alpha_1 \rightarrow \alpha_2\} \cup \{\alpha_2 \dot{=} int\} \cup V(\{x : \alpha_1\}, x, int) \cup V(\{x : \alpha_1\}, 1, int) \\ &= \{\alpha \dot{=} \alpha_1 \rightarrow \alpha_2, \alpha_2 \dot{=} int\} \cup \{\alpha_1 \dot{=} int\} \cup \{int \dot{=} int\} \\ &= \{\alpha \dot{=} \alpha_1 \rightarrow \alpha_2, \alpha_2 \dot{=} int, \alpha_1 \dot{=} int\} \end{aligned}$$

Unification (2)

$$y = 11 - 3x$$

- Solving: by the unification procedure
- Solution = a substitution that satisfies all the collected equations
 - Substitute a type variable to a known type or another type variable
- The mapping $unify(\tau_1, \tau_2)$ from type variables to types makes τ_1 equivalent to τ_2

$$unify(\alpha, \tau) = \{\alpha \mapsto \tau\} \quad \text{if } \alpha \notin \tau$$

$$unify(\tau, \alpha) = \{\alpha \mapsto \tau\} \quad \text{if } \alpha \notin \tau$$

$$\begin{aligned} unify(\tau_1 \rightarrow \tau_2, \tau'_1 \rightarrow \tau'_2) = & \text{let } S_1 = unify(\tau_1, \tau'_1) \\ & S_2 = unify(S\tau_2, S\tau'_2) \\ & \text{in } S_2 S_1 \end{aligned}$$

$$unify(\tau_1, \tau'_1) = failure \quad \text{other cases}$$

Example

$$y = 11 - 3x$$

- $unify(\alpha, int) = \{\alpha \mapsto int\}$
- $unify(\alpha_1 \rightarrow \alpha_2, int \rightarrow int) = let\ S_1 = unify(\alpha_1, int)\ and\ S_2 = unify(S_1\ \alpha_2, S_1\ int)$
 $in\ S_2\ S_1$
 $= let\ S_1 = \{\alpha_1 \mapsto int\}\ and\ S_2 = unify(S_1\ \alpha_2, S_1\ int)$
 $in\ S_2\ S_1$
 $= let\ S_1 = \{\alpha_1 \mapsto int\}\ and\ S_2 = unify(\alpha_2, int)$
 $in\ S_2\ S_1$
 $= let\ S_1 = \{\alpha_1 \mapsto int\}\ and\ S_2 = \{\alpha_2 \mapsto int\}$
 $in\ S_2\ S_1$
 $= \{\alpha_2 \mapsto int\} \circ \{\alpha_1 \mapsto int\}$

Unification (3)

$$2x + (11 - 3x) = 8$$

- Final solution: a simple accumulation of the substitution

$$\text{Solve}(\{\tau_1 \doteq \tau_2\}) = \text{unify}(\tau_1, \tau_2)$$

$$\begin{aligned} \text{Solve}(\{\tau_1 \doteq \tau_2\} \cup \text{rest}) = & \text{let } S = \text{unify}(\tau_1, \tau_2) \\ & \text{in } (\text{Solve}(S \text{ rest})) S \end{aligned}$$

- For a program E , the type inference is

$$\text{Solve}(V(\emptyset, E, \alpha))$$

$$\begin{aligned} x &= 3 \\ y &= 2 \end{aligned}$$

Theorem. Let E be a program and α a fresh type variable. S is a solution for the collection $V(\emptyset, E, \alpha)$ of type equations if and only if judgement $\emptyset \vdash E : S \alpha$ is provable.

Example

- $\lambda x. x + 1$

$$\begin{aligned} V(\emptyset, \lambda x. x + 1, \alpha) &= \{\alpha \dot{=} \alpha_1 \rightarrow \alpha_2\} \cup V(\{x : \alpha_1\}, x + 1, \alpha_2) \\ &= \{\alpha \dot{=} \alpha_1 \rightarrow \alpha_2\} \cup \{\alpha_2 \dot{=} int\} \cup V(\{x : \alpha_1\}, x, int) \cup V(\{x : \alpha_1\}, 1, int) \\ &= \{\alpha \dot{=} \alpha_1 \rightarrow \alpha_2, \alpha_2 \dot{=} int\} \cup \{\alpha_1 \dot{=} int\} \cup \{int \dot{=} int\} \\ &= \{\alpha \dot{=} \alpha_1 \rightarrow \alpha_2, \alpha_2 \dot{=} int, \alpha_1 \dot{=} int\} \end{aligned}$$

$$\begin{aligned} Solve(\{\alpha \dot{=} \alpha_1 \rightarrow \alpha_2, \alpha_2 \dot{=} int, \alpha_1 \dot{=} int\}) &= let\ S = unify(\alpha, \alpha_1 \rightarrow \alpha_2) \\ &\quad in\ (Solve(S\ rest))\ S \\ &= \{\alpha_1 \mapsto int\} \circ \{\alpha_2 \mapsto int\} \circ \{\alpha \mapsto (\alpha_1 \rightarrow \alpha_2)\} \\ &= \{\alpha \mapsto (int \rightarrow int)\} \end{aligned}$$

Conclusion

- Type systems: a specialized framework of static analysis
 - Static analysis by proof construction
- Unification algorithm: an efficient algorithm for type inference
 - No fixed point iterations but simple substitutions