Program Analysis

16. Static Analysis by Proof Construction

Kihong Heo

KAIST

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:

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

- Simple types:

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

- 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 Rules

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

$$\frac{\Gamma \vdash E_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash E_2 : \tau_1}{\Gamma \vdash E_1\ E_2 : \tau_2} \qquad \frac{\Gamma \vdash E_1 : int \qquad \Gamma \vdash E_2 : int}{\Gamma \vdash E_1 + E_2 : int}$$

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:

$$
\dfrac{\dfrac{\dfrac{\dfrac{x : int \in \{x : int\}}{\{x : int\} \vdash x : int} \qquad \{x : int\} \vdash 1 : int}{\{x : int\} \vdash x + 1 : int}}{\emptyset \vdash \lambda x.x + 1 : int \to int} \qquad \emptyset \vdash 2 : int}{\emptyset \vdash (\lambda x.x + 1)(2) : int}
$$

Soundness & Expressiveness

- Sound type system:
  - The program is provable = 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

$$
\begin{aligned}
V(\Gamma, n, \tau) &= \{\tau \doteq int\} \\
V(\Gamma, x, \tau) &= \{\tau \doteq \Gamma(x)\} \\
V(\Gamma, \lambda x.E, \tau) &= \{\tau \doteq \alpha_1 \to \alpha_2\} \cup V(\Gamma + x : \alpha_1, E, \alpha_2) && (\text{new } \alpha_i) \\
V(\Gamma, E_1 E_2, \tau) &= V(\Gamma, E_1, \alpha \to \tau) \cup V(\Gamma, E_2, \alpha) && (\text{new } \alpha) \\
V(\Gamma, E_1 + E_2, \tau) &= \{\tau \doteq int\} \cup V(\Gamma, E_1, int) \cup V(\Gamma, E_2, int)
\end{aligned}
$$

Example

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

- $\lambda x \,.\, x + 1$

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

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

$$
\begin{array}{rcll}
unify(\alpha, \tau) & = & \{\alpha \mapsto \tau\} & \text{if } \alpha \notin \tau \\
unify(\tau, \alpha) & = & \{\alpha \mapsto \tau\} & \text{if } \alpha \notin \tau \\
unify(\tau_1 \rightarrow \tau_2, \tau_1' \rightarrow \tau_2') & = & let\ \ S_1 = unify(\tau_1, \tau_1') & \\
 & & \quad\quad\ \ S_2 = unify(S_1\tau_2, S_1\tau_2') & \\
 & & in\ \ S_2\ S_1 & \\
unify(\tau_1, \tau_1') & = & failure & \text{other cases}
\end{array}
$$

Example

$y = 11 - 3x$

- $unify(\alpha, int) = \{\alpha \mapsto int\}$

- $$\begin{aligned} unify(\alpha_1 \to \alpha_2, int \to int) &= let\ S_1 = unify(\alpha_1, int) \text{ and } S_2 = unify(S_1\ \alpha_2, S_1\ int) \\ &\quad in\ S_2\ S_1 \\ &= let\ S_1 = \{\alpha_1 \mapsto int\} \text{ and } S_2 = unify(S_1\ \alpha_2, S_1\ int) \\ &\quad in\ S_2\ S_1 \\ &= let\ S_1 = \{\alpha_1 \mapsto int\} \text{ and } S_2 = unify(\alpha_2, int) \\ &\quad in\ S_2\ S_1 \\ &= let\ S_1 = \{\alpha_1 \mapsto int\} \text{ and } S_2 = \{\alpha_2 \mapsto int\} \\ &\quad in\ S_2\ S_1 \\ &= \{\alpha_2 \mapsto int\} \circ \{\alpha_1 \mapsto int\} \end{aligned}$$

Unification (3)

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

- Final solution: a simple accumulation of the substitution

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

- For a program E, the type inference is

$$Solve(V(\emptyset, E, \alpha))$$

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

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

Example

- $\lambda x\,.\,x + 1$

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

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

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