

Overview

Week 10 - Types

Summary of Week 9

Core ML

Type Checking

Type Inference

Overview

Week 10 - Types

Summary of Week 9

Core ML

Type Checking

Type Inference

Core ML

Definition (Expressions)

$$e := \underbrace{x \mid e \ e \mid \lambda x. e}_{\lambda\text{-Calculus}} \mid \underbrace{c}_{\text{primitives/constants}} \mid \underbrace{\text{let } x = e \text{ in } e}_{\text{let binding}} \mid \underbrace{\text{if } e \text{ then } e \text{ else } e}_{\text{conditional}}$$

Primitives

Boolean: true, false, <, >, ...

Arithmetic: ×, +, ÷, −, 0, 1, ...

Tuples: pair, fst, snd

Lists: nil, cons, hd, tl

What is Type Checking?

Given some **environment** (assigning types to primitives) together with a core ML **expression** and a **type**, check whether the expression really has that type with respect to the environment.

Preliminaries

Definition

Types

- ▶ **type variables** $\alpha, \alpha_0, \alpha_1, \dots$
- ▶ **arrow type constructor** $'\rightarrow'$
- ▶ **type constructors** g, g_1, \dots (like: list)
- ▶ **type** τ

$$\tau ::= \alpha \mid \tau \rightarrow \tau \mid g(\tau, \dots, \tau)$$

- ▶ special case - **base types**: int, bool (instead of int(), bool())

The Type Checking System \mathcal{C}

$$\frac{}{E, e : \tau \vdash e : \tau} \text{ (ref)} \quad \frac{E \vdash e_1 : \tau_2 \rightarrow \tau_1 \quad E \vdash e_2 : \tau_2}{E \vdash e_1 e_2 : \tau_1} \text{ (app)}$$

$$\frac{E, x : \tau_1 \vdash e : \tau_2}{E \vdash \lambda x. e : \tau_1 \rightarrow \tau_2} \text{ (abs)} \quad \frac{E \vdash e_1 : \tau_1 \quad E, x : \tau_1 \vdash e_2 : \tau_2}{E \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{ (let)}$$

$$\frac{E \vdash e_1 : \text{bool} \quad E \vdash e_2 : \tau \quad E \vdash e_3 : \tau}{E \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau} \text{ (ite)}$$

Preliminaries (cont'd)

(Typing) **environment**: set of pairs E mapping variables and primitives to types (instead of $(e, \tau) \in E$ write $(e : \tau) \in E$, i.e., “ e is of type τ in E ”)

Domain: of typing environment E

$$\text{Dom}(E) = \{e \mid (e : \tau) \in E\}$$

(Typing) **judgment**: $E \vdash e : \tau$ states “it can be **proved** that expression e has type τ in environment E ”

Example

- ▶ primitive environment
 $P = \{+ : \text{int} \rightarrow \text{int} \rightarrow \text{int}, \text{nil} : \text{list}(\alpha), \text{true} : \text{bool}, \dots\}$
- ▶ $\text{Dom}(P) = \{+, \text{nil}, \text{true}, \dots\}$
- ▶ $P \vdash \text{true} : \text{bool}$

Example

- ▶ environment $E = \{\text{true} : \text{bool}, + : \text{int} \rightarrow \text{int} \rightarrow \text{int}\}$
- ▶ judgment $E \vdash (\lambda x. x) \text{ true} : \text{bool}$

Proof.

$$\frac{E, \{x : \text{bool}\} \vdash x : \text{bool}}{E \vdash \lambda x. x : \text{bool} \rightarrow \text{bool}} \text{ (abs)} \quad \frac{E \vdash \lambda x. x : \text{bool} \rightarrow \text{bool} \quad E \vdash \text{true} : \text{bool}}{E \vdash (\lambda x. x) \text{ true} : \text{bool}} \text{ (app)}$$

□

Example

- ▶ environment $E = \{\text{true} : \text{bool}, + : \text{int} \rightarrow \text{int} \rightarrow \text{int}\}$
- ▶ judgment $E \vdash \lambda x.x + x : \text{int} \rightarrow \text{int}$

Proof.

Blackboard



What is Type Inference?

Given some **environment** together with a core ML **expression** and a **type**, infer a **solution** (type substitution)—if possible—such that the **most general type** of the expression is obtained.

Overview

Week 10 - Types

Summary of Week 9

Core ML

Type Checking

Type Inference

Preliminaries

Type substitution: σ is mapping from type variables to types

Application:

$$\tau\sigma \stackrel{\text{def}}{=} \begin{cases} \sigma(\alpha) & \text{if } \tau = \alpha \\ \tau_1\sigma \rightarrow \tau_2\sigma & \text{if } \tau = \tau_1 \rightarrow \tau_2 \\ g(\tau_1\sigma, \dots, \tau_n\sigma) & \text{if } \tau = g(\tau_1, \dots, \tau_n) \end{cases}$$

$$E\sigma \stackrel{\text{def}}{=} \{e : \tau\sigma \mid e : \tau \in E\}$$

Type variables:

$$\mathcal{TVar}(\tau) \stackrel{\text{def}}{=} \begin{cases} \{\alpha\} & \text{if } \tau = \alpha \\ \mathcal{TVar}(\tau_1) \cup \mathcal{TVar}(\tau_2) & \text{if } \tau = \tau_1 \rightarrow \tau_2 \\ \bigcup_{1 \leq i \leq n} \mathcal{TVar}(\tau_i) & \text{if } \tau = g(\tau_1, \dots, \tau_n) \end{cases}$$

Composition: $\sigma_1\sigma_2 \stackrel{\text{def}}{=} \sigma_2 \circ \sigma_1$

Unification Problems

Definition

- ▶ equation $\tau \approx \tau'$ is **satisfiable** if exists σ s.t., $\tau\sigma = \tau'\sigma$
- ▶ σ is called **solution** of $\tau \approx \tau'$
- ▶ **unification problem** is (finite) sequence of equations

$$\tau_1 \approx \tau'_1; \dots; \tau_n \approx \tau'_n$$

- ▶ \square denotes **empty sequence**
- ▶ solving given unification problem is called **unification**

Example

$$\text{list}(\text{bool}) \approx \text{list}(\alpha) \begin{matrix} \Rightarrow_l^{(d_1)} & \text{bool} \approx \alpha \\ \Rightarrow_{\{\alpha \mapsto \text{bool}\}}^{(v_2)} & \square \end{matrix}$$

The Unification System \mathcal{U}

$$\frac{E_1; g(\tau_1, \dots, \tau_n) \approx g(\tau'_1, \dots, \tau'_n); E_2}{E_1; \tau_1 \approx \tau'_1; \dots; \tau_n \approx \tau'_n; E_2} \text{ (d}_1\text{)}$$

$$\frac{E_1; \tau_1 \rightarrow \tau_2 \approx \tau'_1 \rightarrow \tau'_2; E_2}{E_1; \tau_1 \approx \tau'_1; \tau_2 \approx \tau'_2; E_2} \text{ (d}_2\text{)}$$

$$\frac{E_1; \alpha \approx \tau; E_2 \quad \alpha \notin \text{TVar}(\tau) \text{ and } \sigma = \{\alpha \mapsto \tau\}}{(E_1; E_2)\sigma} \text{ (v}_1\text{)}$$

$$\frac{E_1; \tau \approx \alpha; E_2 \quad \alpha \notin \text{TVar}(\tau) \text{ and } \sigma = \{\alpha \mapsto \tau\}}{(E_1; E_2)\sigma} \text{ (v}_2\text{)}$$

$$\frac{E_1; \tau \approx \tau; E_2}{E_1; E_2} \text{ (t)}$$

Type Inference Problems

- ▶ $E \triangleright e : \tau$ is **type inference problem**
- ▶ σ s.t., $E\sigma \vdash e : \tau\sigma$ (if exists) is **solution** (otherwise: e not typable)

The Type Inference System \mathcal{I}

$$\frac{E, e : \tau_0 \triangleright e : \tau_1}{\tau_0 \approx \tau_1} \text{ (con)} \qquad \frac{E \triangleright e_1 \ e_2 : \tau}{E \triangleright e_1 : \alpha \rightarrow \tau; E \triangleright e_2 : \alpha} \text{ (app)}$$

$$\frac{E \triangleright \lambda x. e : \tau}{E, x : \alpha_1 \triangleright e : \alpha_2; \tau \approx \alpha_1 \rightarrow \alpha_2} \text{ (abs)} \qquad \frac{E \triangleright \mathbf{let} \ x = e_1 \ \mathbf{in} \ e_2 : \tau}{E \triangleright e_1 : \alpha; E, x : \alpha \triangleright e_2 : \tau} \text{ (let)}$$

$$\frac{E \triangleright \mathbf{if} \ e_1 \ \mathbf{then} \ e_2 \ \mathbf{else} \ e_3 : \tau}{E \triangleright e_1 : \mathbf{bool}; E \triangleright e_2 : \tau; E \triangleright e_3 : \tau} \text{ (ite)}$$

Example

find most general type of $\mathbf{let} \ id = \lambda x. x \ \mathbf{in} \ id \ 1$ w.r.t. P

Recipe - Type Inference

Input

core ML expression e and typing environment E

Algorithm

1. generate $E \triangleright e : \alpha$ (fresh type variable α)
2. use \mathcal{I} to transform $E \triangleright e : \alpha$ to unification problem u (if at any point no rule applicable **Not Typable**)
3. if possible solve u (obtaining unifier σ) otherwise **Not Typable**

Output

the **most general** type of e w.r.t. E is $\alpha\sigma$