Functional Programming
WS 2007/08

Christian Sternagel[1] (VO + PS)
Friedrich Neurauter[2] (PS)
Harald Zankl[3] (PS)

Computational Logic
Institute of Computer Science

University of Innsbruck

11 January 2008

[1]christian.sternagel@uibk.ac.at
[2]friedrich.neurauter@uibk.ac.at
[3]harald.zankl@uibk.ac.at

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

Combinator Parsing

Notes

- decompose linear sequence (text) into structure (type)
- type 'a Parser.t is Strng.t -> 'a result
- primitive parser sat : (char -> bool) -> char Parser.t
- primitive combinators (>>=) [bind], (<|>) [choice], return, many
- problem: left recursion

Overview

Week 10 - Types

- Summary of Week 9
- Core ML
- Type Checking
- Type Inference

Core ML

Definition (Expressions)

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

Primitives

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

Arithmetic: $\times$, $+$, $\div$, $-$, 0, 1, ...

Tuples: pair, fst, snd

Lists: nil, cons, hd, tl

Overview

Week 10 - Types

Summary of Week 9

Core ML

Type Checking

Type Inference

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 α, α_0, α_1, ...
- arrow type constructor '$\rightarrow$'
- type constructors g, g_1, ... (like: list)
- type τ

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

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

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

$$\mathcal{D}\text{om}(E) = \{e \mid (e : \tau) \in E\}$$

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

Example

- primitive environment
  $P = \{+ : \text{int} \to \text{int} \to \text{int}, \text{nil} : \text{list}(\alpha), \text{true} : \text{bool}, \ldots\}$
- $\mathcal{D}\text{om}(P) = \{+, \text{nil}, \text{true}, \ldots\}$
- $P \vdash \text{true} : \text{bool}$

The Type Checking System $\mathcal{C}$

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

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

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

Example

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

Proof.

$$\dfrac{\dfrac{E, \{x : \mathsf{bool}\} \vdash x : \mathsf{bool}}{E \vdash \lambda x.x : \mathsf{bool} \to \mathsf{bool}}\,(\mathsf{abs}) \qquad E \vdash \mathsf{true} : \mathsf{bool}}{E \vdash (\lambda x.x)\ \mathsf{true} : \mathsf{bool}}\,(\mathsf{app})$$

□

Example

- environment $E = \{\mathsf{true} : \mathsf{bool}, + : \mathsf{int} \to \mathsf{int} \to \mathsf{int}\}$
- judgment $E \vdash \lambda x.x + x : \mathsf{int} \to \mathsf{int}$

Proof.

Blackboard □

Overview

Week 10 - Types
- Summary of Week 9
- Core ML
- Type Checking
- Type Inference

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.

Preliminaries

Type substitution: σ is mapping from type variables to types

Application:

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

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

Type variables:

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

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

$$\tau_1 \approx \tau_1'; \ldots; \tau_n \approx \tau_n'$$

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

The Unification System $\mathcal{U}$

$$\frac{E_1; g(\tau_1, \ldots, \tau_n) \approx g(\tau_1', \ldots, \tau_n'); E_2}{E_1; \tau_1 \approx \tau_1'; \ldots; \tau_n \approx \tau_n'; E_2}\ (\mathsf{d}_1)$$

$$\frac{E_1; \tau_1 \to \tau_2 \approx \tau_1' \to \tau_2'; E_2}{E_1; \tau_1 \approx \tau_1'; \tau_2 \approx \tau_2'; E_2}\ (\mathsf{d}_2)$$

$$\frac{E_1; \alpha \approx \tau; E_2 \quad \alpha \notin \mathcal{TV}\mathrm{ar}(\tau) \text{ and } \sigma = \{\alpha \mapsto \tau\}}{(E_1; E_2)\sigma}\ (\mathsf{v}_1)$$

$$\frac{E_1; \tau \approx \alpha; E_2 \quad \alpha \notin \mathcal{TV}\mathrm{ar}(\tau) \text{ and } \sigma = \{\alpha \mapsto \tau\}}{(E_1; E_2)\sigma}\ (\mathsf{v}_2)$$

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

Example

$$\begin{aligned} \mathsf{list}(\mathsf{bool}) \approx \mathsf{list}(\alpha) \quad &\Rightarrow_{\iota}^{(\mathsf{d}_1)} \quad \mathsf{bool} \approx \alpha \\ &\Rightarrow_{\{\alpha \mapsto \mathsf{bool}\}}^{(\mathsf{v}_2)} \quad \square \end{aligned}$$

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 $\mathcal{I}$

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

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

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

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 $\alpha\sigma$

Example

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