Functional Programming

WS 2009/10

Christian Sternagel (VO)
Friedrich Neurauter (PS) Sarah Winkler (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

11 December 2009

Combinator Parsing

Notes

- decompose linear sequence (text) into structure (type)
- type `('a,'t)Parser.t` is
  `'t list -> ('a * 't list)option`
- primitive parser
  `token : ('t -> 'a option) -> ('a,'t)Parser.t`
- primitive combinators $\underbrace{\texttt{(>>=)}}_{\text{bind}}$, $\underbrace{\texttt{(<|>)}}_{\text{choice}}$, `return`, `many`

This Week

Practice I

OCaml introduction, lists, strings, trees

Theory I

lambda-calculus, evaluation strategies, induction, reasoning about functional programs

Practice II

efficiency, tail-recursion, combinator-parsing

Theory II

type checking, type inference

Advanced Topics

lazy evaluation, infinite data structures, monads, ...

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

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")

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

Example

- primitive environment
  $P = \{+ : \text{int} \rightarrow \text{int} \rightarrow \text{int}, \text{NIL} : \text{list}(\alpha), \text{TRUE} : \text{bool}, \ldots\}$
- $P \vdash \text{TRUE} : \text{bool}$

The Type Checking System $\mathcal{C}$

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

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

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

Example

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

Proof.

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

□

Example

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

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.

Preliminaries

Type substitution: σ is mapping from type variables to types

Application:

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

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

Type variables:

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

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 $\mathcal{U}$

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

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

$$\frac{E_1; \alpha \approx \tau; E_2 \quad \alpha \notin \mathcal{TV}\mathsf{ar}(\tau)}{(E_1; E_2)\{\alpha/\tau\}} \ (\mathrm{v}_1)$$

$$\frac{E_1; \tau \approx \alpha; E_2 \quad \alpha \notin \mathcal{TV}\mathsf{ar}(\tau)}{(E_1; E_2)\{\alpha/\tau\}} \ (\mathrm{v}_2)$$

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

Example

$$\begin{array}{lll} \mathsf{list}(\mathsf{bool}) \approx \mathsf{list}(\alpha) & \Rightarrow_{\iota}^{(\mathsf{d}_1)} & \mathsf{bool} \approx \alpha \\ & \Rightarrow_{\{\alpha/\mathsf{bool}\}}^{(\mathsf{v}_2)} & \square \end{array}$$

Type Inference Problems

- $E \rhd e : \tau$ 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 $\mathcal{I}$

$$\frac{E, e : \tau_0 \rhd e : \tau_1}{\tau_0 \approx \tau_1} \text{ (con)}$$

$$\frac{E \rhd e_1\ e_2 : \tau}{E \rhd e_1 : \alpha \to \tau; E \rhd e_2 : \alpha} \text{ (app)}$$

$$\frac{E \rhd \lambda x.e : \tau}{E, x : \alpha_1 \rhd e : \alpha_2; \tau \approx \alpha_1 \to \alpha_2} \text{ (abs)}$$

$$\frac{E \rhd \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau}{E \rhd e_1 : \alpha; E, x : \alpha \rhd e_2 : \tau} \text{ (let)}$$

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

Recipe - Type Inference

Input

core ML expression e and typing environment E

Algorithm

1. generate $E \rhd e : \alpha$ (fresh type variable α)
2. use $\mathcal{I}$ to transform $E \rhd e : \alpha$ 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

Proof.

Blackboard □