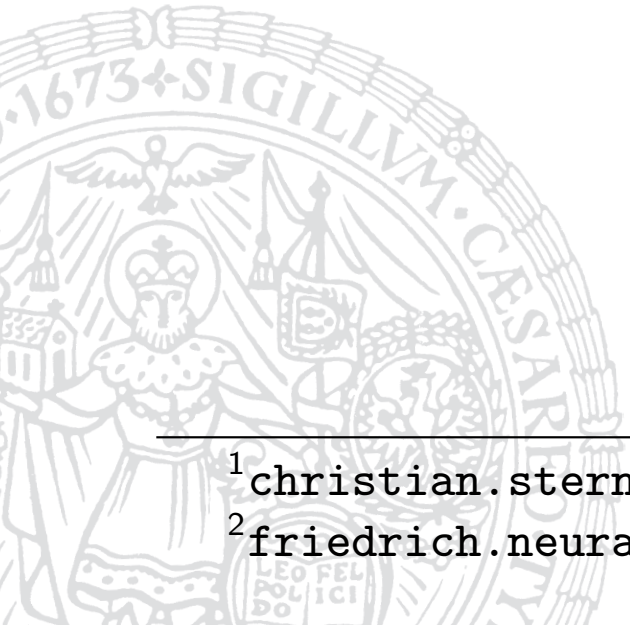
Functional Programming

WS 2008/09

Christian Sternagel (VO)[1] Friedrich Neurauter (PS)[2]

Computational Logic
Institute of Computer Science
University of Innsbruck

12 December 2008



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

Type Checking

Problem

$$\overbrace{E}^{\text{Environment}} \vdash \underbrace{e}_{\text{expression}} : \overbrace{\tau}^{\text{Type}}$$

Does e have type τ under E?

Solution

A proof tree using the inference rules of $\mathcal{C}$.

Type Inference

Problem

$$E \rhd e : \tau$$

Is there a substitution σ such that $E\sigma \vdash e : \tau\sigma$ holds?

Solution

1. Transform $E \rhd e : \tau$ into a unification problem using the inference rules of $\mathcal{I}$.
2. Solve the unification problem using the inference rules of $\mathcal{U}$.

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

Grammar

$e ::= x \mid c \mid (e) \mid e \llcorner e \mid \lambda x.e \mid \textbf{let}\ x = e\ \textbf{in}\ e \mid \textbf{if}\ e\ \textbf{then}\ e\ \textbf{else}\ e$

Removing Left Recursion

$e ::= xf \mid cf \mid (e)f \mid \lambda x.ef \mid \textbf{let}\ x = e\ \textbf{in}\ ef \mid \textbf{if}\ e\ \textbf{then}\ e\ \textbf{else}\ ef$

$f ::= \llcorner ef \mid \epsilon$

Making Application Left Associative

$e ::= gf$

$f ::= \llcorner gf \mid \epsilon$

$g ::= x \mid c \mid (e) \mid \lambda x.e \mid \textbf{let}\ x = e\ \textbf{in}\ e \mid \textbf{if}\ e\ \textbf{then}\ e\ \textbf{else}\ e$

A Type for Core ML Expressions

```
type t = Var of Strng.t
       | Con of Strng.t
       | App of (t * t)
       | Abs of (Strng.t * t)
       | Let of (Strng.t * t * t)
       | Ite of (t * t * t)
```

Core ML Expressions from Strings

```
of_string : string -> t
```

Recall

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

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

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

A Type for Types

Grammar

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

```
type typ = TVar of int
         | TFun of (typ * typ)
         | TCon of (Strng.t * typ list)
```

Data Structures

Input

- environment: `type env = (CoreML.t * typ) list`
- type inference problem:
  `type ip = (env * CoreML.t * typ)`

Output

unification problem `type up = (typ * typ) list`

Function

`to_up : ip -> up`

Recall

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

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

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

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

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

Data Structures

Input

unification problem `type up = (typ * typ) list`

Output

substitution `type sub = (int * typ) list`

Function

```
unify : up -> sub
```

Type Inference

```
let infer s =
 let e  = CoreML.of_string s in
 let up = to_up(pmu,e,tvar 1) in
 let s  = unify up in
 sub s (tvar 1)
```