Functional Programming

WS 2014/15

Cezary Kaliszyk (VO+PS)
Yann Savoye (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

week 11

Overview

- Week 11 - Implementing Type Inference
  - Summary of Week 10
  - A Module for Core ML Expressions
  - Implementing $\mathcal{I}$
  - Implementing $\mathcal{U}$

Overview

- Week 11 - Implementing Type Inference
  - Summary of Week 10
  - A Module for Core ML Expressions
  - Implementing $\mathcal{I}$
  - Implementing $\mathcal{U}$

Type Checking

Problem

$$E \vdash e : \tau$$

Does e have type τ under E?

Type Checking

Problem

$$\overbrace{E}^{\text{Environment}} \vdash \; e \; : \; \tau$$

Does e have type τ under E?

Type Checking

Problem

$$E \vdash \underbrace{e}_{\text{expression}} : \tau$$

Does e have type τ under E?

Type Checking

Problem

$$E \vdash e : \overbrace{\tau}^{\text{Type}}$$

Does e have type τ under E?

Type Checking

Problem

$$E \vdash e : \tau$$

Does e have type τ under E?

Solution

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

Type Inference

Problem

$$E \rhd e : \alpha_0$$

Is there a substitution σ *such that* $E\sigma \vdash e : \alpha_0\sigma$ *holds?*

Type Inference

Problem

$$E \rhd e : \alpha_0$$

Is there a substitution σ such that $E\sigma \vdash e : \alpha_0\sigma$ holds?

Solution

Type Inference

Problem

$$E \rhd e : \alpha_0$$

Is there a substitution σ such that $E\sigma \vdash e : \alpha_0\sigma$ holds?

Solution

1. Transform $E \rhd e : \alpha_0$ into a unification problem using the inference rules of $\mathcal{I}$.

Type Inference

Problem

$$E \rhd e : \alpha_0$$

Is there a substitution σ such that $E\sigma \vdash e : \alpha_0\sigma$ holds?

Solution

1. Transform $E \rhd e : \alpha_0$ into a unification problem using the inference rules of $\mathcal{I}$.
2. Solve the unification problem using the inference rules of $\mathcal{U}$.

Overview

- Week 11 - Implementing Type Inference
  - Summary of Week 10
  - A Module for Core ML Expressions
  - Implementing $\mathcal{I}$
  - Implementing $\mathcal{U}$

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, dynamic programming

Theory II

type checking, type inference

Advanced Topics

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

Overview

- Week 11 - Implementing Type Inference
  - Summary of Week 10
  - A Module for Core ML Expressions
  - Implementing $\mathcal{I}$
  - Implementing $\mathcal{U}$

Core ML

Grammar

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

Core ML

Grammar

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

Core ML in OCaml (module CoreML)

```
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 (cont'd)

Magic

- Remove Left Recursion
- Make Application Left Associative
- Binding Precedence
- `CoreML.of_string : string -> CoreML.t`

Core ML (cont'd)

Magic

- Remove Left Recursion
- Make Application Left Associative
- Binding Precedence
- `CoreML.of_string : string -> CoreML.t`

Example

```
# CoreML.of_string "\\x.(+ x) x";;
- : CoreML.t = \x.+ x x
# CoreML.of_string "\\x.+ (x x)";;
- : CoreML.t = \x.+ (x x)
```

Overview

- Week 11 - Implementing Type Inference
  - Summary of Week 10
  - A Module for Core ML Expressions
  - Implementing $\mathcal{I}$
  - Implementing $\mathcal{U}$

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 (module Typing)

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)
```

A Type for Types (module Typing)

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)

let tvar v = TVar v

let (@->) s t = TFun(s,t)

let tcon c ts = TCon(Strng.of_string c,ts)
```

Data Structures (module `Typing`)

Input

Data Structures (module Typing)

Input

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

Data Structures (module Typing)

Input

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

Data Structures (module Typing)

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`

Data Structures (module Typing)

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
```

```
let to_up_step i (env,e,t) = match Lst.lookup e env with
 | Some t' -> (i,[(t',t)],[])
 | None -> match e with
  | App(e1,e2)    ->
   (i+1,[],[(env,e1,tvar i @-> t);(env,e2,tvar i)])
  | Abs(x,e)      ->
   (i+2,[(t,tvar i @-> tvar(i+1))],
        [((Var x,tvar i)::env,e,tvar(i+1))])
  | Let(x,e1,e2)  ->
   (i+1,[],[(env,e1,tvar i);((Var x,tvar i)::env,e2,t)])
  | Ite(e1,e2,e3) ->
   (i,[],[(env,e1,tbool);(env,e2,t);(env,e3,t)])
  | Var x         -> failwith("unknown `"^Strng.to_string x^"'")
```

```
let to_up (env,e,t) =
 let rec to_up (i,eqs) = function
  | [] -> (i,eqs)
  | p::ps ->
   let (i,eqs2,qs) = to_up_step i p in
   let (i,eqs1) = to_up (i,eqs) qs in
   to_up (i,eqs1@eqs2) ps
 in
 let i = (*largest type variable occurring in t and env*)
  Lst.foldr (fun (_,t) -> max (max_tvar t)) (max_tvar t) env in
 snd (to_up (i+1,[]) [(env,e,t)])
```

Overview

- Week 11 - Implementing Type Inference
  - Summary of Week 10
  - A Module for Core ML Expressions
  - Implementing $\mathcal{I}$
  - Implementing $\mathcal{U}$

Recall

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

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

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

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

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

Data Structures

Input

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

Data Structures

Input

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

Output

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

Data Structures

Input

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

Output

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

Function

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

```
let unify eqs =
 let rec unify s = function
  | [] -> s
  | eq::eqs ->
   let (e,s') = step eq in
   let eqs' = Lst.map (fun(l,r) -> (sub s' l,sub s' r)) eqs in
   unify (s' <*> s) (e @ eqs')
 in
 unify [] eqs

let (<*>) sub2 sub1 = (* sub2 after sub1 *)
 let d1 = dom sub1 in
 Lst.map (fun (a,t) -> (a,sub sub2 t)) sub1
  @ Lst.filter (fun (a,_) -> not(Lst.mem a d1)) sub2
```

```
let step = function
 | (s,t) when s = t              -> ([],[])
 | (TVar a,t) | (t,TVar a)       ->
  if St.mem a (tvars t) then failwith "occur␣check!"
                        else ([],[(a,t)])
 | (TFun(s1,t1),TFun(s2,t2)) -> ([(s1,s2);(t1,t2)],[])
 | (TCon(g,ss),TCon(h,ts))       ->
  if g = h then (Lst.zip ss ts,[])
           else failwith("mismatch:␣‘"^(Strng.to_string g)
                         ^"’␣vs.␣‘"^(Strng.to_string h)^"’")
```

Type Inference

```
let infer s =
 let e  = CoreML.of_string s in
 let up = to_up(pmu,e,tvar 0) in
 let s  = unify up in
 sub s (tvar 0)
```