universität innsbruck

WS 2018/2019

Functional Programming

Lecture 10

Cezary Kaliszyk Jonas Schöpf
Christian Sternagel Vincent van Oostrom

Department of Computer Science

Topics

abstract data types, algebraic data types, binary search trees, combinator parsing, efficiency, encoding data types as lambda-terms, evaluation strategies, formal verification, first steps, guarded recursion, Haskell introduction, higher-order functions, historical overview, induction, infinite data structures, input and output, lambda-calculus, lazy evaluation, list comprehensions, lists, modules, pattern matching, polymorphism, property-based testing, reasoning about functional programs, recursive functions, sets, strings, tail recursion, trees, tupling, type checking, type inference, types, types and type classes, unification, user-defined types

Overview

- Type Checking
- Unification and its Implementation
- Type Inference

Problem – Type Checking

input: expression e and type τ

output: YES (e has type τ) or NO

Haskell Examples

```
(\x -> x) True :: Bool
```

```
double :: Int -> Int
double x = x + x
```

The Language of Expressions – Core FP

$$
\begin{array}{rcll}
e & ::= & x \mid e\ e \mid \lambda x.\, e & \text{λ-terms} \\
  & \mid & c & \text{constant (for primitives)} \\
  & \mid & \textbf{let}\ x = e\ \textbf{in}\ e & \text{let-binding} \\
  & \mid & \textbf{if}\ e\ \textbf{then}\ e\ \textbf{else}\ e & \text{conditional branching}
\end{array}
$$

Primitives

- used for predefined “functions” and “constants”
- **Boolean:** True, False, $<$, $>$, $=$, $\ldots$
- **arithmetic:** $\times$, $+$, $\div$, $-$, 0, 1, $\ldots$
- **tuples:** Pair, fst, snd
- **lists:** Nil, Cons, head, tail

Types

- type variables α, α_0, α_1, ...
- (right-associative) function space constructor $\to$
- type constructors C, C_0, C_1, ... (like List for type of lists)
- types $\tau ::= \alpha \mid \tau \to \tau \mid C(\tau, \ldots, \tau)$
- special case – base types: Int, Bool (instead of Int(), Bool())

Example Types

- List(Bool) – list of Booleans
- Pair(Int, Int) – pairs of integers
- Int $\to$ Int $\to$ Bool – functions from two integers to Booleans

A Type for Types

```
data Type = TVar Int              -- for easy renaming of type variables
    | TCon String [Type]          -- type constructors and function type
  deriving Eq                     -- type equality can be checked
```

Typing Environments

- set of pairs E, mapping variables and constants to types
- instead of $(e, \tau) \in E$, write $e :: \tau \in E$

Typing Judgments

- $E \vdash e :: \tau$
- read: "it can be proved that e is of type τ under E"

Examples

- primitive environment
  $P = \{\mathsf{True} :: \mathsf{Bool}, + :: \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}, \mathsf{Nil} :: \mathsf{List}(\alpha), \ldots\}$
- $P \vdash \mathsf{True} :: \mathsf{Bool}$ – "using primitive environment, it can be shown that True is of type Bool"

Type Substitutions

- mapping σ from type variables to types
- apply substitution to type

$$\alpha\sigma = \sigma(\alpha)$$
$$(\tau_1 \to \tau_2)\sigma = \tau_1\sigma \to \tau_2\sigma$$
$$C(\tau_1, \ldots, \tau_n)\sigma = C(\tau_1\sigma, \ldots, \tau_n\sigma)$$

- composition $\sigma_1\sigma_2 = (\lambda x.\, \sigma_1(x)\sigma_2)$ ("first apply σ_1 and then σ_2")

Examples

domain

- $\sigma_1 = \{\alpha_1 \mapsto \mathsf{List}(\alpha_2), \alpha_2 \mapsto \mathsf{Bool}\}$
- $\tau = \mathsf{List}(\alpha_1)$
- $\tau\sigma_1 = \mathsf{List}(\mathsf{List}(\alpha_2))$
- $\sigma_2 = \{\alpha_2 \mapsto \mathsf{Int}, \alpha_3 \mapsto \mathsf{Int}\}$
- $\sigma_1\sigma_2 = \{\alpha_1 \mapsto \mathsf{List}(\mathsf{Int}), \alpha_2 \mapsto \mathsf{Bool}, \alpha_3 \mapsto \mathsf{Int}\}$

Implementing Type Substitutions

```
type TSub = [(Int, Type)]
```

"Extracting" Maybes with Default

- `Data.Maybe.fromMaybe :: a -> Maybe a -> a`
- satisfies equations $\texttt{fromMaybe}\ x\ \texttt{Nothing} = x$

  $\texttt{fromMaybe}\ x\ (\texttt{Just}\ y) = y$

Applying Substitutions to Types

```
tsub :: TSub -> Type -> Type
tsub s (x@(TVar i)) = fromMaybe x $ lookup i s
tsub s (TCon g ts)  = TCon g (map (tsub s) ts)
```

Composition

```
tcomp :: TSub -> TSub -> TSub
s1 `tcomp` s2 =
    map (\(x, t) -> (x, tsub s2 t)) s1 ++
      filter ((`notElem` dom1) . fst) s2
  where dom1 = map fst s1
```

Type Checking as Natural Deduction Rules

$$\frac{e :: \tau \in E}{e :: \tau\sigma}\ (\mathsf{ins}) \qquad \frac{e_1 :: \tau_2 \to \tau_1 \qquad e_2 :: \tau_2}{e_1\ e_2 :: \tau_1}\ (\mathsf{app})$$

$$\frac{\boxed{\begin{array}{c} x :: \tau_1 \\ \vdots \\ e :: \tau_2 \end{array}}}{\lambda x.\, e :: \tau_1 \to \tau_2}\ (\mathsf{abs}) \qquad \frac{e_1 :: \tau_1 \qquad \boxed{\begin{array}{c} x :: \tau_1 \\ \vdots \\ e_2 :: \tau_2 \end{array}}}{\textbf{let}\ x = e_1\ \textbf{in}\ e_2 :: \tau_2}\ (\mathsf{let})$$

$$\frac{e :: \tau}{e :: \tau}\ (\mathsf{copy}) \qquad \frac{e_1 :: \mathsf{Bool} \qquad e_2 :: \tau \qquad e_3 :: \tau}{\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 :: \tau}\ (\mathsf{ite})$$

Examples

- environment $E = \{\mathsf{True} :: \mathsf{Bool}, + :: \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}\}$
- recall `(\x -> x) True :: Bool`
- prove judgment $E \vdash (\lambda x.\, x)$ True :: Bool
-

1	True :: Bool	ins E
2	x :: Bool	assumption
3	$\lambda x.\, x$ :: Bool $\to$ Bool	abs 2
4	$(\lambda x.\, x)$ True :: Bool	app 3, 1

- recall

```
double :: Int -> Int
double x = x + x
```

- prove $E \vdash \lambda x.\, x + x$ :: Int $\to$ Int
-

1	x :: Int	assumption
2	$+$ :: Int $\to$ Int $\to$ Int	ins E
3	$(+)\ x$ :: Int $\to$ Int	app 2, 1
4	$x + x$ :: Int	app 3, 1
5	$\lambda x.\, x + x$ :: Int $\to$ Int	abs 1–4

Problem – Unification

pair of types

input: equation $\tau_1 \approx \tau_2$

syntactic equality

output: most general unifier (mgu) σ with $\tau_1\sigma = \tau_2\sigma$ or FAILURE

a type substitution

Notions

- equation $\tau \approx \tau'$ is satisfiable iff exists σ such that $\tau\sigma = \tau'\sigma$
- if $\tau \approx \tau'$ satisfiable by σ, then 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
- unification – solving given unification problem
- set of type variables of a type

$$\mathcal{V}(\alpha) = \{\alpha\}$$
$$\mathcal{V}(\tau_1 \to \tau_2) = \mathcal{V}(\tau_1) \cup \mathcal{V}(\tau_2)$$
$$\mathcal{V}(C(\tau_1, \ldots, \tau_n)) = \mathcal{V}(\tau_1) \cup \ldots \cup \mathcal{V}(\tau_n)$$

Unification Rules

(d) decomposition

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

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

(t) removal of trivial equations

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

(v) variable elimination

$$\frac{E_1; \alpha \approx \tau; E_2 \quad \alpha \notin \mathcal{V}(\tau)}{(E_1; E_2)\{\alpha \mapsto \tau\}} \; (\mathsf{v}_1) \qquad \frac{E_1; \tau \approx \alpha; E_2 \quad \alpha \notin \mathcal{V}(\tau)}{(E_1; E_2)\{\alpha \mapsto \tau\}} \; (\mathsf{v}_2)$$

Applying Unification Rules

- notation: $E \Rightarrow_{\sigma}^{(r)} E'$
- meaning: apply rule r to derive E' from E using substitution σ
- (if r is not variable elimination, σ is empty substitution)
- resulting substitution of sequence

$$E_1 \Rightarrow_{\sigma_1}^{(r_1)} E_2 \Rightarrow_{\sigma_2}^{(r_2)} E_3 \Rightarrow_{\sigma_3}^{(r_3)} \ldots \Rightarrow_{\sigma_n}^{(r_n)} E_{n+1}$$

is composition

$$\sigma_1 \sigma_2 \cdots \sigma_n$$

Example

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

mgu: $\{\}\{\alpha \mapsto \mathsf{Bool}\} = \{\alpha \mapsto \mathsf{Bool}\}$

`Unification.hs` – Implementing Unification

Interface

- input: unification problem `type UP = [(Type, Type)]`
- output: type substitution
- unification: `unify :: UP -> TSub`

The Free Variables of a Type

```
tvars :: Type -> [Int]
tvars (TVar x)     = [x]
tvars (TCon g ts) = foldr (union . tvars) [] ts
```

Applying Type Substitutions to Unification Problems

```
tsubUP :: TSub -> UP -> UP
tsubUP s = map (\(l, r) -> (tsub s l, tsub s r))
```

Unification

```
unify :: UP -> TSub
unify eqs = go [] eqs
  where
    go s []       = s
    go s (eq:eqs) =
      go (s `tcomp` s') (eqs' ++ tsubUP s' eqs)
      where
        (eqs', s') = step eq
    step :: (Type, Type) -> (UP, TSub)
    ...
```

Removal of Trivial Equations

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

```
step (t, u) | t == u = ([], [])
```

Variable Elimination

$$\frac{E_1; \alpha \approx \tau; E_2 \quad \alpha \notin \mathcal{V}(\tau)}{(E_1; E_2)\{\alpha \mapsto \tau\}}\ (\mathsf{v}_1) \qquad \frac{E_1; \tau \approx \alpha; E_2 \quad \alpha \notin \mathcal{V}(\tau)}{(E_1; E_2)\{\alpha \mapsto \tau\}}\ (\mathsf{v}_2)$$

```
step (TVar x, t) = singletonSub x t
step (t, TVar x) = singletonSub x t
  where
    notUnif = error "not unifiable"
    singletonSub x t =
      if x `elem` tvars t then notUnif else ([], [(x, t)])
```

Decomposition

$$\frac{E_1; C(\tau_1, \ldots, \tau_n) \approx C(\tau_1', \ldots, \tau_n'); E_2}{E_1; \tau_1 \approx \tau_1'; \ldots; \tau_n \approx \tau_n'; E_2}\ (\mathsf{d}_1) \quad \frac{E_1; \tau_1 \to \tau_2 \approx \tau_1' \to \tau_2'; E_2}{E_1; \tau_1 \approx \tau_1'; \tau_2 \approx \tau_2'; E_2}\ (\mathsf{d}_2)$$

```
step (TCon f ts, TCon g us) |
  f == g && length ts == length us = (zip ts us, [])
```

Type Inference Problems

- $E \rhd e :: \tau$
- read: "try to infer most general substitution σ such that $E \vdash e :: \tau\sigma$"

Haskell Example

what is most general type of `let id = \x -> x in id 0`

Example

- $E = \{0 :: \mathsf{Int}\}$
- $E \rhd \textbf{let } id = \lambda x.\, x \textbf{ in } id\ 0 :: \alpha_0$
- $\sigma = \{\alpha_0 \mapsto \mathsf{Int}\}$
-

1	$x :: \mathsf{Int}$	assumption
2	$\lambda x.\, x :: \mathsf{Int} \to \mathsf{Int}$	abs 1
3	$id :: \mathsf{Int} \to \mathsf{Int}$	assumption
4	$0 :: \mathsf{Int}$	ins E
5	$id\ 0 :: \mathsf{Int}$	app 3, 4
6	$\textbf{let } id = \lambda x.\, x \textbf{ in } id\ 0 :: \mathsf{Int}$	let 2, 3–5

Typing Constraint Rules

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

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

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

where α in (app) and (let), and α_1 and α_2 in (abs) are fresh (do not occur in premises of rule)

Recipe – Type Inference

- to find most general type of e under E
- first take $E \rhd e :: \alpha_0$ (for some fresh type variable α_0)
- then, use typing constraint rules to generate unification problem u (if at any point no rule applicable Not Typable)
- if u has no solution (none of the unification rules is applicable before reaching $\square$) then Not Typable, otherwise, obtain solution σ
- finally, $\alpha_0\sigma$ is most general type of e

Exercise

- find most general type of **let** $id = \lambda x.\, x$ **in** $id\ 1$ with respect to P
- start from $P \triangleright$ **let** $id = \lambda x.\, x$ **in** $id\ 1 :: \alpha_0$

$$\underline{P \triangleright \textbf{let}\ id = \lambda x.\, x\ \textbf{in}\ id\ 1 :: \alpha_0}$$

$$\Rightarrow^{(\mathsf{let})}\ \underline{P \triangleright \lambda x.\, x :: \alpha_1}; \ P, id :: \alpha_1 \triangleright id\ 1 :: \alpha_0$$

$$\Rightarrow^{(\mathsf{abs})}\ \alpha_1 \approx \alpha_2 \to \alpha_3; \ \underline{P, x :: \alpha_2 \triangleright x :: \alpha_3}; \ P, id :: \alpha_1 \triangleright id\ 1 :: \alpha_0$$

$$\Rightarrow^{(\mathsf{con})}\ \alpha_1 \approx \alpha_2 \to \alpha_3; \alpha_2 \approx \alpha_3; \ \underline{P, id :: \alpha_1 \triangleright id\ 1 :: \alpha_0}$$

$$\Rightarrow^{(\mathsf{app})}\ \alpha_1 \approx \alpha_2 \to \alpha_3; \alpha_2 \approx \alpha_3; \ \underline{P, id :: \alpha_1 \triangleright id :: \alpha_4 \to \alpha_0}; \ \underline{P, id :: \alpha_1 \triangleright 1 :: \alpha_4}$$

$$\Rightarrow^{(\mathsf{con})^2}\ \alpha_1 \approx \alpha_2 \to \alpha_3; \alpha_2 \approx \alpha_3; \ \alpha_1 \approx \alpha_4 \to \alpha_0; \mathsf{Int} \approx \alpha_4$$

$$\alpha_1 \approx \alpha_2 \to \alpha_3; \underline{\alpha_2 \approx \alpha_3}; \ \alpha_1 \approx \alpha_4 \to \alpha_0; \underline{\mathsf{Int} \approx \alpha_4}$$

$$\Rightarrow^{(\mathsf{v})^2}_{\{\alpha_2 \mapsto \alpha_3, \alpha_4 \mapsto \mathsf{Int}\}}\ \underline{\alpha_1 \approx \alpha_3 \to \alpha_3}; \alpha_1 \approx \mathsf{Int} \to \alpha_0$$

$$\Rightarrow^{(\mathsf{v})}_{\{\alpha_1 \mapsto \alpha_3 \to \alpha_3\}}\ \underline{\alpha_3 \to \alpha_3 \approx \mathsf{Int} \to \alpha_0}$$

$$\Rightarrow^{(\mathsf{d})}\ \underline{\alpha_3 \approx \mathsf{Int}}; \underline{\alpha_3 \approx \alpha_0}$$

$$\Rightarrow^{(\mathsf{v})^2}_{\{\alpha_0 \mapsto \mathsf{Int}, \alpha_3 \mapsto \mathsf{Int}\}}\ \square$$

mgu: $\{\alpha_0 \mapsto \mathsf{Int}, \alpha_1 \mapsto \mathsf{Int} \to \mathsf{Int}, \alpha_2 \mapsto \mathsf{Int}, \alpha_3 \mapsto \mathsf{Int}, \alpha_4 \mapsto \mathsf{Int}\}$

Exams

1st Exam

- 12:15–14:00 in HSB 1 on February 1, 2019
- online registration required before 23:59 on January 18, 2019

2nd Exam

- 12:15–14:00 in HSB 1 on March 1, 2019
- online registration required before 23:59 on February 15, 2019

3rd Exam

- 12:15–14:00 in HSB 1 on September 27, 2019
- online registration required before 23:59 on September 13, 2019

Some Old Exams

- exam of February 2, 2018 + solutions
- exam of March 2, 2018 + solutions

Homework (for January 25th)

1. Read the lecture notes about type checking and type inference.
2. Use type checking to prove $\{f :: \alpha \to \beta, y :: \alpha\} \vdash (\lambda x.\, f\ x)\ y :: \beta$.
3. Solve the unification problem
   $\alpha_2 \approx \alpha_1 \to \alpha_0; \alpha_3 \approx \mathsf{Int} \to \alpha_2; \alpha_2 \approx \alpha_3$
   and compute the resulting mgu, if possible. Check your result using `unify`.
4. Solve the unification problem $\mathsf{Pair}(\mathsf{Bool}, \alpha_0) \approx \mathsf{Pair}(\alpha_1, \mathsf{Int})$. Check your solution using `unify`.
5. Use type inference to obtain the most general type of the expression $map\ (\lambda x.\, x)$ with respect to the environment
   $E = \{map :: (\alpha \to \beta) \to \mathsf{List}(\alpha) \to \mathsf{List}(\beta)\}$.
6. Given the following types for environments and inference problems
   ```
   type Env = [(String, Type)]
   type IP = [(Env, Exp, Type)]
   ```
   implement a function `toUp :: IP -> UP` that transforms a given type inference problem into the corresponding unification problem according to our typing constraint rules.