

Functional Programming

Christian Sternagel Harald Zankl Evgeny Zuenko

Department of Computer Science
University of Innsbruck

WS 2017/2018

Lecture 10



Overview

- Type Checking
- Unification
- Type Inference

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, **implementing a type checker**, 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

Problem – Type Checking

Type Checking

input: expression e and type τ
output: YES (e has type τ) or NO

The Language of Expressions – Core FP

$e ::= x$	$e e$	$\lambda x. e$	λ -terms
	c		constant (for primitives)
	let $x = e$ in e		let-binding
	if e then e else e		conditional branching

Primitives

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

What is Type Checking?

Given a (*typing*) *environment* (mapping variables to types), a *core FP expression*, and a *type*, check whether the expression is of the given type with respect to the environment.

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**
 $P = \{\text{True} :: \text{Bool}, + :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}, \text{Nil} :: \text{List}(\alpha), \dots\}$
- $P \vdash \text{True} :: \text{Bool}$ – “using the primitive environment, it can be shown that True is of type Bool”

Types

- **type variables**, $\alpha, \alpha_0, \alpha_1, \dots$
- the (right-associative) **function space constructor** \rightarrow
- **type constructors** C, C_1, \dots (like List, for the type of lists)
- **types** $\tau ::= \alpha \mid \tau \rightarrow \tau \mid C(\tau, \dots, \tau)$
- special case – **base types**: Int, Bool (instead of Int(), Bool())

Example Types

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

Type Substitutions

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

$$\begin{aligned}\alpha\sigma &= \sigma(\alpha) \\ (\tau_1 \rightarrow \tau_2)\sigma &= \tau_1\sigma \rightarrow \tau_2\sigma \\ C(\tau_1, \dots, \tau_n)\sigma &= C(\tau_1\sigma, \dots, \tau_n\sigma)\end{aligned}$$

- **composition** $\sigma_1\sigma_2 = (\lambda x. \sigma_1(x)\sigma_2)$ (“first apply σ_1 and then σ_2 ”)

Examples

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

Type Checking as Natural Deduction Rules

$$\frac{e :: \tau \in E}{e :: \tau\sigma} \text{ (ins)}$$

$$\frac{e_1 :: \tau_2 \rightarrow \tau_1 \quad e_2 :: \tau_2}{e_1 e_2 :: \tau_1} \text{ (app)}$$

$$\frac{\boxed{\begin{array}{c} x :: \tau_1 \\ \vdots \\ e :: \tau_2 \end{array}}}{\lambda x. e :: \tau_1 \rightarrow \tau_2} \text{ (abs)}$$

$$\frac{e_1 :: \tau_1 \quad \boxed{\begin{array}{c} x :: \tau_1 \\ \vdots \\ e_2 :: \tau_2 \end{array}}}{\text{let } x = e_1 \text{ in } e_2 :: \tau_2} \text{ (let)}$$

$$\frac{e :: \tau}{e :: \tau} \text{ (copy)}$$

$$\frac{e_1 :: \text{Bool} \quad e_2 :: \tau \quad e_3 :: \tau}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 :: \tau} \text{ (ite)}$$

Problem – Unification

pair of types

Unification

input: equation $\tau_1 \approx \tau_2$

syntactic equality

output: most general unifier σ such that $\tau_1\sigma = \tau_2\sigma$ or FAILURE

Notions

a type substitution

- equation $\tau \approx \tau'$ is **satisfiable** iff exists σ such that $\tau\sigma = \tau'\sigma$
- σ is called **solution** of $\tau \approx \tau'$
- **unification problem** is finite sequence of equations

$$\tau_1 \approx \tau'_1; \dots; \tau_n \approx \tau'_n$$

- \square denotes empty sequence
- **unification** – solving given unification problem
- **set of type variables** of a type

$$\mathcal{V}(\alpha) = \{\alpha\}$$

$$\mathcal{V}(\tau_1 \rightarrow \tau_2) = \mathcal{V}(\tau_1) \cup \mathcal{V}(\tau_2)$$

$$\mathcal{V}(C(\tau_1, \dots, \tau_n)) = \mathcal{V}(\tau_1) \cup \dots \cup \mathcal{V}(\tau_n)$$

Examples

- environment $E = \{\text{True} :: \text{Bool}, + :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}\}$
- prove judgment $E \vdash (\lambda x. x) \text{ True} :: \text{Bool}$
- 1 True :: Bool ins E
- 2 $x :: \text{Bool}$ assumption
- 3 $\lambda x. x :: \text{Bool} \rightarrow \text{Bool}$ abs 2
- 4 $(\lambda x. x) \text{ True} :: \text{Bool}$ app 3, 1
- prove $E \vdash \lambda x. x + x :: \text{Int} \rightarrow \text{Int}$
- 1 $x :: \text{Int}$ assumption
- 2 $+ :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$ ins E
- 3 $(+) x :: \text{Int} \rightarrow \text{Int}$ app 2, 1
- 4 $x + x :: \text{Int}$ app 3, 1
- 5 $\lambda x. x + x :: \text{Int} \rightarrow \text{Int}$ abs 1–4

Unification Rules

Unification

(d) **decomposition**

$$\frac{E_1; C(\tau_1, \dots, \tau_n) \approx C(\tau'_1, \dots, \tau'_n); E_2}{E_1; \tau_1 \approx \tau'_1; \dots; \tau_n \approx \tau'_n; E_2} \text{ (d}_1\text{)}$$

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

(t) removal of **trivial** equations

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

(v) **variable** elimination

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

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

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

$$E_1 \Rightarrow_{\sigma_1}^{(r_1)} E_2 \Rightarrow_{\sigma_2}^{(r_2)} E_3 \Rightarrow_{\sigma_3}^{(r_3)} \dots \Rightarrow_{\sigma_{n-1}}^{(r_{n-1})} E_n$$

is composition

$$\sigma_1 \sigma_2 \dots \sigma_{n-1}$$

Example

$$\begin{aligned} \text{List}(\text{Bool}) \approx \text{List}(\alpha) &\Rightarrow_{\{\}}^{(d_1)} \text{Bool} \approx \alpha \\ &\Rightarrow_{\{\alpha \mapsto \text{Bool}\}}^{(v_2)} \square \end{aligned}$$

Type Inference Problems

- $E \triangleright e :: \tau$
- read: “try to infer most general substitution σ such that $E \vdash e :: \tau\sigma$ ”

Example

- $E = \{0 :: \text{Int}\}$
- $E \triangleright \text{let } id = \lambda x. x \text{ in } id \ 0 :: \alpha_0$
- $\sigma = \{\alpha_0 \mapsto \text{Int}\}$

1	$x :: \text{Int}$	assumption
2	$\lambda x. x :: \text{Int} \rightarrow \text{Int}$	abs 1
3	$id :: \text{Int} \rightarrow \text{Int}$	assumption
4	$0 :: \text{Int}$	ins E
5	$id \ 0 :: \text{Int}$	app 3, 4
6	$\text{let } id = \lambda x. x \text{ in } id \ 0 :: \text{Int}$	let 2, 3-5

What is Type Inference?

Given an *environment* (assigning types to primitives), a core FP expression, and an initial type, try to infer a *solution* (that is, type substitution) such that applying it to the initial type yields the *most general compatible type* of the initial expression.

Typing Constraint Rules

$$\frac{E, e :: \tau_0 \triangleright e :: \tau_1}{\tau_0 \approx \tau_1} \text{ (con)} \quad \frac{E \triangleright e_1 \ e_2 :: \tau}{E \triangleright e_1 :: \alpha \rightarrow \tau; E \triangleright e_2 :: \alpha} \text{ (app)}$$

$$\frac{E \triangleright \lambda x. e :: \tau}{\tau \approx \alpha_1 \rightarrow \alpha_2; E, x :: \alpha_1 \triangleright e :: \alpha_2} \text{ (abs)} \quad \frac{E \triangleright \text{let } x = e_1 \text{ in } e_2 :: \tau}{E \triangleright e_1 :: \alpha; E, x :: \alpha \triangleright e_2 :: \tau} \text{ (let)}$$

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

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 $E \triangleright e :: \alpha_0$ (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 $P \triangleright \mathbf{let}\ id = \lambda x. x\ \mathbf{in}\ id\ 1 :: \alpha_0$

$$\begin{array}{l}
 \Rightarrow^{(\text{let})} \frac{P \triangleright \mathbf{let}\ id = \lambda x. x\ \mathbf{in}\ id\ 1 :: \alpha_0}{P \triangleright \lambda x. x :: \alpha_1;} \quad \alpha_1 \approx \alpha_2 \rightarrow \alpha_3; \alpha_2 \approx \alpha_3; \\
 \frac{P, id :: \alpha_1 \triangleright id\ 1 :: \alpha_0}{\alpha_1 \approx \alpha_4 \rightarrow \alpha_0; \mathbf{Int} \approx \alpha_4} \\
 \Rightarrow^{(\text{abs})} \frac{\alpha_1 \approx \alpha_2 \rightarrow \alpha_3;}{P, x :: \alpha_2 \triangleright x :: \alpha_3;} \quad \Rightarrow^{(v)^2} \frac{\alpha_1 \approx \alpha_4 \rightarrow \alpha_0; \mathbf{Int} \approx \alpha_4}{\{\alpha_2 \mapsto \alpha_3, \alpha_4 \mapsto \mathbf{Int}\}} \\
 \frac{P, id :: \alpha_1 \triangleright id\ 1 :: \alpha_0}{\alpha_1 \approx \alpha_3 \rightarrow \alpha_3; \alpha_1 \approx \mathbf{Int} \rightarrow \alpha_0} \\
 \Rightarrow^{(\text{con})} \frac{\alpha_1 \approx \alpha_2 \rightarrow \alpha_3; \alpha_2 \approx \alpha_3;}{P, id :: \alpha_1 \triangleright id\ 1 :: \alpha_0} \quad \Rightarrow^{(v)} \frac{\alpha_1 \approx \alpha_3 \rightarrow \alpha_3; \alpha_1 \approx \mathbf{Int} \rightarrow \alpha_0}{\{\alpha_1 \mapsto \alpha_3 \rightarrow \alpha_3\}} \\
 \frac{\alpha_3 \rightarrow \alpha_3 \approx \mathbf{Int} \rightarrow \alpha_0}{\Rightarrow^{(d)}} \\
 \Rightarrow^{(\text{app})} \frac{\alpha_1 \approx \alpha_2 \rightarrow \alpha_3; \alpha_2 \approx \alpha_3;}{P, id :: \alpha_1 \triangleright id :: \alpha_4 \rightarrow \alpha_0;} \quad \frac{\alpha_3 \approx \mathbf{Int}; \alpha_3 \approx \alpha_0}{\Rightarrow^{(v)^2} \{\alpha_0 \mapsto \mathbf{Int}, \alpha_3 \mapsto \mathbf{Int}\}} \\
 \frac{P, id :: \alpha_1 \triangleright 1 :: \alpha_4}{\alpha_1 \approx \alpha_2 \rightarrow \alpha_3; \alpha_2 \approx \alpha_3;} \quad \square \\
 \Rightarrow^{(\text{con})^2} \frac{\alpha_1 \approx \alpha_2 \rightarrow \alpha_3; \alpha_2 \approx \alpha_3;}{\alpha_1 \approx \alpha_4 \rightarrow \alpha_0; \mathbf{Int} \approx \alpha_4} \quad \text{mgu: } \{\alpha_0 \mapsto \mathbf{Int}, \alpha_1 \mapsto \mathbf{Int} \rightarrow \mathbf{Int}, \\
 \alpha_2 \mapsto \mathbf{Int}, \alpha_3 \mapsto \mathbf{Int}, \alpha_4 \mapsto \mathbf{Int}\}
 \end{array}$$

Exercises (for January 19th)

1. Read the lecture notes about type checking and type inference.
2. Check that **if** True **then** $x + 1$ **else** $x - 1$ is of type \mathbf{Int} under $P \cup \{x :: \mathbf{Int}\}$.
3. Give a proof of $\emptyset \vdash \lambda xy. x :: \alpha_0 \rightarrow \alpha_1 \rightarrow \alpha_0$.
4. Solve the unification problem $\text{Pair}(\mathbf{Bool}, \alpha_0) \approx \text{Pair}(\alpha_1, \mathbf{Int})$.
5. Show that the unification problem $\text{Pair}(\mathbf{Bool}, \alpha_0) \approx \text{Pair}(\alpha_0, \mathbf{Int})$ does not have a solution.
6. Infer the most general type of **let** $suc = \lambda x. x + 1$ **in** **let** $d = \lambda x. suc\ (suc\ x)$ **in** $d\ 2$ under P .