Exercises of the lecture on static typing

Programming Paradigms · Session 10 · exercises of the lecture slides

Exercise $m.n$ is exercise n of the slide "exercises" of block m; exercises 1.8 and 2.5 are stated on the slides "several possible types" and "a remark". Each system is recalled in a box before its exercises. A *derivation* of a judgment is a tree: the judgment stands at the root, every node is the conclusion of an instance of one of the rules, and the node's children are the premises of that instance.

Block 1: the simply typed λ-calculus

The simply typed λ-calculus, with a type annotation on each binder

Terms	$e ::= n \mid b \mid x \mid \lambda x{:}\tau.\, e \mid e\ e \mid e\ op\ e$	n an integer, $b \in \{\texttt{true}, \texttt{false}\}$
Types	$\tau ::= \texttt{int} \mid \texttt{bool} \mid \tau \to \tau$	$op \in \{+, <\}$
Judgment	$\Gamma \vdash e : \tau$	Γ a function from variables to types

$$\frac{}{\Gamma \vdash n : \texttt{int}}\ \text{(T-Int)} \qquad \frac{}{\Gamma \vdash b : \texttt{bool}}\ \text{(T-Bool)} \qquad \frac{x \in \mathrm{dom}(\Gamma)}{\Gamma \vdash x : \Gamma(x)}\ \text{(T-Var)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}\ \text{(T-Plus)} \qquad \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 < e_2 : \texttt{bool}}\ \text{(T-Less)}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash (\lambda x{:}\tau_1.\, e) : \tau_1 \to \tau_2}\ \text{(T-Abs)} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ \text{(T-App)}$$

Conventions. $\Gamma, x{:}\tau$ is Γ in which x now has the type τ (an earlier type of x is hidden); $\emptyset$ is the empty environment, and $x{:}\tau$ alone stands for $\emptyset, x{:}\tau$. The arrow associates to the right: $\texttt{int} \to \texttt{int} \to \texttt{bool}$ is $\texttt{int} \to (\texttt{int} \to \texttt{bool})$. Application associates to the left and binds tighter than $+$, which binds tighter than $<$. The body of an abstraction extends as far to the right as possible.

The calculus without annotations. Its terms are $e ::= n \mid b \mid x \mid \lambda x.\, e \mid e\ e \mid e\ op\ e$, with the same types. Evaluation ignores the annotations: $(\lambda x{:}\tau.\, e)\ v \to e\,[x := v]$.

Exercise 1.1 Two typing derivations

Write typing derivation of

$$\emptyset \vdash (\lambda x{:}\texttt{bool}.\, x)\ \texttt{true} : \texttt{bool} \qquad \text{and} \qquad \emptyset \vdash \lambda x{:}\texttt{int}.\, \lambda y{:}\texttt{int}.\, x < y : \texttt{int} \to \texttt{int} \to \texttt{bool}.$$

Exercise 1.2 Two annotated identity functions

Are $(\lambda x{:}\texttt{bool}.\, x)$ and $(\lambda x{:}\texttt{int}.\, x)$ the same programs?

Exercise 1.3 Typing rules without annotations

Redesign typing rules for the λ-calculus without explicit type parameters. What is the type of the identity function $\lambda x.\, x$ now?

Exercise 1.4 A term without annotations

Write typing derivation for $\emptyset \vdash \lambda x.\, \lambda y.\, x\ y : \tau$, for a type τ to find.

Exercise 1.5 Erasure of type annotations

Define type-annotation erasure function $\texttt{erase}(\cdot)$ and state its correctness theorem.

Exercise 1.6 Typability in the empty context

What does it mean for a program e to be well-typed in an empty context, i.e. $\emptyset \vdash e : \tau$ for some type τ?

Exercise 1.7 Self-application

Is there any context Γ and type τ such that $\Gamma \vdash x\ x : \tau$?

Block 1: mini-ML with monomorphic typing

Mini-ML. A function $\texttt{fun}\ x \to e$ carries no annotation, and a primitive stands before its argument: $+(x, 1)$.

$$\begin{array}{lll} \text{Expressions} & e ::= x \mid c \mid op \mid \texttt{fun}\ x \to e \mid e\ e \mid (e, e) \mid \texttt{let}\ x = e\ \texttt{in}\ e & \\ & c \text{ a constant } (1, 2, \ldots, \mathit{true}, \ldots), \quad op \text{ a primitive } (+, \times, \mathit{fst}, \ldots) & \\ \text{Types} & \tau ::= \texttt{int} \mid \texttt{bool} \mid \cdots \mid \tau \to \tau \mid \tau \times \tau & \end{array}$$

$$\frac{x \in \mathrm{dom}(\Gamma)}{\Gamma \vdash x : \Gamma(x)}\ (\textsc{Var}) \qquad \frac{}{\Gamma \vdash n : \texttt{int}}\ (\textsc{Const})\ \ldots \qquad \frac{}{\Gamma \vdash + : \texttt{int} \times \texttt{int} \to \texttt{int}}\ (\textsc{Prim})\ \ldots$$

$$\frac{\Gamma + x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \texttt{fun}\ x \to e : \tau_1 \to \tau_2}\ (\textsc{Fun}) \qquad \frac{\Gamma \vdash e_1 : \tau' \to \tau \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\ e_2 : \tau}\ (\textsc{App})$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}\ (\textsc{Pair}) \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma + x{:}\tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2}\ (\textsc{Let})$$

$\Gamma + x{:}\tau$ is the environment Γ' defined by $\Gamma'(x) = \tau$ and $\Gamma'(y) = \Gamma(y)$ otherwise. The slides give these rules without names; the names are added here, to label the steps of a derivation.

Exercise 1.8 An identity function applied to itself

Show that the expression

$$(\texttt{fun}\ x \to x)\ (\texttt{fun}\ x \to x)\ \ 42$$

is typable.

Block 2: the typing of the While language

The While language (written WHILE on the slides)

Expressions	$e ::= c \mid x \mid e \; op \; e$
	c an integer or Boolean constant, op a binary operator $(+, <, \dots)$
Statements	$s ::= x \leftarrow e \mid \texttt{if}\ e\ \texttt{then}\ s\ \texttt{else}\ s \mid \texttt{while}\ e\ \texttt{do}\ s \mid s;\ s \mid \texttt{skip}$
Types	$\tau ::= \texttt{int} \mid \texttt{bool}$

Typing of expressions: $\Gamma \vdash e : \tau$, "in environment Γ, expression e has type τ".

$$\frac{}{\Gamma \vdash n : \texttt{int}}\ \text{(T-Int)} \qquad \frac{}{\Gamma \vdash b : \texttt{bool}}\ \text{(T-Bool)} \qquad \frac{x \in \mathrm{dom}(\Gamma)}{\Gamma \vdash x : \Gamma(x)}\ \text{(T-Var)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}\ \text{(T-Plus)} \qquad \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 < e_2 : \texttt{bool}}\ \text{(T-Less)} \qquad \text{etc.}$$

Type checking of statements: $\Gamma \vdash s$, "in environment Γ, statement s is well-typed".

$$\frac{}{\Gamma \vdash \texttt{skip}}\ \text{(T-Skip)} \qquad \frac{\Gamma \vdash s_1 \quad \Gamma \vdash s_2}{\Gamma \vdash s_1; s_2}\ \text{(T-Seq)} \qquad \frac{\Gamma \vdash x : \tau \quad \Gamma \vdash e : \tau}{\Gamma \vdash x \leftarrow e}\ \text{(T-Assign)}$$

$$\frac{\Gamma \vdash e : \texttt{bool} \quad \Gamma \vdash s_1 \quad \Gamma \vdash s_2}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2}\ \text{(T-If)} \qquad \frac{\Gamma \vdash e : \texttt{bool} \quad \Gamma \vdash s}{\Gamma \vdash \texttt{while}\ e\ \texttt{do}\ s}\ \text{(T-While)}$$

The slides give these rules without names; the names are added here, to label the steps of a derivation.

Consider a small extension of the WHILE language with

a) new expressions: pair constructors (e, e) and get the left and the right pair components written $\texttt{fst}\, e$ and $\texttt{snd}\, e$ respectively;
b) new statements: $\texttt{for}\ x = e\ \texttt{to}\ e\ \texttt{do}\ s$ and $\texttt{repeat}\ s\ \texttt{until}\ e$ loops.

Exercise 2.1 Pair types and the typing of pairs

Define the pair types. Define typing rules for each of the pair (e_1, e_2), $\texttt{fst}\, e$ and $\texttt{snd}\, e$ expressions.

Exercise 2.2 A rejected program

Consider the program

$$\texttt{if fst}\, x > 0\ \texttt{then}\ x \leftarrow (\texttt{snd}\, x, \texttt{fst}\, x + \texttt{snd}\, x)\ \texttt{else}\ x \leftarrow 0$$

Explain why this program is rejected by the static type system.

Exercise 2.3 The typing rule of the repeat loop

Define the typing rule for the repeat loop $\texttt{repeat}\ s\ \texttt{until}\ e$.

Exercise 2.4 The typing rule of the for loop

Define the typing rule for the for loop $\texttt{for}\ i = e_1\ \texttt{to}\ e_2\ \texttt{do}\ s$.

Exercise 2.5 The conditional as a statement and as an expression

The WHILE language has the conditional at the level of statements (the rule on the left); a conditional could be added in the expressions (the rule on the right):

$$\frac{\Gamma \vdash e : \texttt{bool} \quad \Gamma \vdash s_1 \quad \Gamma \vdash s_2}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2} \qquad \frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

What is different and what is common in both cases?