Bordeaux University
Year 2015-2016

Master ESSV
MtoCPL

TD n°2 - Type inference

Exercice 1: Basic types

Starting from the previous language, Pierce defines the following typing rules :

Syntax :

$\mathsf{T} ::=$		*types*
	Bool	*type of booleans*

Typing rules :

$$\mathtt{true} : \mathsf{Bool}$$

$$\mathtt{false} : \mathsf{Bool}$$

$$[\textsc{If}]\ \frac{\mathsf{t}_1 : \mathsf{Bool} \qquad \mathsf{t}_2 : \mathsf{T}_1 \qquad \mathsf{t}_3 : \mathsf{T}_1}{\mathtt{if}\ \mathsf{t}_1\ \mathtt{then}\ \mathsf{t}_2\ \mathtt{else}\ \mathsf{t}_3 : \mathsf{T}_1}$$

Syntax :

$\mathsf{T} ::=$	$\ldots$	*types*
	Nat	*type of naturals*

Typing rules :

$$\mathtt{0} : \mathsf{Nat}$$

$$[\textsc{Succ}]\ \frac{\mathsf{t}_1 : \mathsf{Nat}}{\mathtt{succ}\ \mathsf{t}_1 : \mathsf{Nat}}$$

$$[\textsc{IsZero}]\ \frac{\mathsf{t}_1 : \mathsf{Nat}}{\mathtt{iszero}\ \mathsf{t}_1 : \mathsf{Bool}}$$

---

1. How to implement types in our language ?
2. What is the role of the type variables T_1 in the [If]-typing rule ?
3. How can we decide whether an expression is correctly typed, and in this case infer the type of this expression ?
4. Write an algorithm that infers a typed derivation tree for a given expression.
5. How would you prove the following results ?
   (i) Each typable expression has at most one type ;
   (ii) There is just one typing derivation (one proof) for checking $t : T$.

Exercice 2: Functions

The natural way to complete our language consists in adding the possibility to write functions :

Syntax :

$\mathsf{t} ::=$	$\ldots$	*terms*
	x	*variable*
	$\lambda \mathsf{x} : \mathsf{T} \,.\, \mathsf{t}$	*abstraction*
	$(\mathsf{t})\ \mathsf{t}$	*application*
$\mathsf{v} ::=$	$\ldots$	*values*
	$\lambda \mathsf{x} : \mathsf{T} \,.\, \mathsf{t}$	*abstraction value*
$\mathsf{T} ::=$	$\ldots$	*types*
	$\mathsf{T} \to \mathsf{T}$	*type of functions*
$\Gamma ::=$		*contexts*
	$\varnothing$	*empty context*
	$\Gamma, \mathsf{x} : \mathsf{T}$	*term binding*

Evaluation rules :

$$\frac{\mathsf{t}_1 \to \mathsf{t}_1'}{(\mathsf{t}_1)\ \mathsf{t}_2 \to (\mathsf{t}_1')\ \mathsf{t}_2}$$

$$\frac{\mathsf{t}_2 \to \mathsf{t}_2'}{(\mathsf{v}_1)\ \mathsf{t}_2 \to (\mathsf{v}_1)\ \mathsf{t}_2'}$$

$$(\lambda \mathsf{x} : \mathsf{T}_1 \,.\, \mathsf{t}_1)\ \mathsf{v}_2 \to [\mathsf{x} \mapsto \mathsf{v}_2]\ \mathsf{t}_1$$

Typing rules :

$$\frac{\mathsf{x} : \mathsf{T}_1 \in \Gamma}{\Gamma \vdash \mathsf{x} : \mathsf{T}_1}$$

$$\frac{\Gamma, \mathsf{x} : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \lambda \mathsf{x} : \mathsf{T}_1 \,.\, \mathsf{t}_2 : \mathsf{T}_1 \to \mathsf{T}_2}$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \to \mathsf{T}_2 \qquad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}_1}{\Gamma \vdash (\mathsf{t}_1)\ \mathsf{t}_2 : \mathsf{T}_2}$$

In this addition, variables make their apparition to denote function parameters, and the notion of context is used for storing a set of typed variables.

1. In which aspects is this language thoroughly different from the previous one ?
2. Write an expression corresponding to the composition of two functions.

Consider the following expression :

```
E  ≡  λ x:X. λ y:Y. λ z:Z. ((x) z) ((y) z) : S
```

3. Find (informally) the set of constraints for E and a substitution that solves them.

Consider the expression for the double function :

```
double  ≡  λ f. λ x. f (f x)
```

Recall that there exist two questions about the typing of this expression :

1. Type Checking : exhibit a derivation tree for this expression using the typing rules. The types need not be the most general, only the tree must be correct with regard to the typing rules. For example :

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{f}:\mathsf{Nat}\to\mathsf{Nat}\in\Gamma}{\Gamma\vdash\mathsf{f}:\mathsf{Nat}\to\mathsf{Nat}}\qquad\dfrac{\dfrac{\mathsf{f}:\mathsf{Nat}\to\mathsf{Nat}\in\Gamma}{\Gamma\vdash\mathsf{f}:\mathsf{Nat}\to\mathsf{Nat}}\quad\dfrac{\mathsf{x}:\mathsf{Nat}\in\Gamma}{\Gamma\vdash\mathsf{x}:\mathsf{Nat}}}{\Gamma\vdash\mathsf{f\,x}:\mathsf{Nat}}}{\Gamma ::= \{\mathsf{f}:\mathsf{Nat}\to\mathsf{Nat},\ \mathsf{x}:\mathsf{Nat}\}\vdash\mathsf{f(f\,x)}:\mathsf{Nat}}}{\{\mathsf{f}:\mathsf{Nat}\to\mathsf{Nat}\}\vdash\lambda\mathsf{x}:\mathsf{Nat}\,.\ \mathsf{f(f\,x)}:\mathsf{Nat}\to\mathsf{Nat}}}{\{\}\vdash\Big(\lambda\,\mathsf{f}:\mathsf{Nat}\to\mathsf{Nat}\,.\ \lambda\mathsf{x}:\mathsf{Nat}\,.\ \mathsf{f(f\,x)}\Big):(\mathsf{Nat}\to\mathsf{Nat})\to\mathsf{Nat}\to\mathsf{Nat}}$$

2. Type Inference : find a type for this expression that is the most general. This is done using for example the typing rules with constraints and solving the constraints :

First pass

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{f}:\mathsf{T}_1\in\Gamma}{\Gamma\vdash\mathsf{f}:\quad}\qquad\dfrac{\dfrac{\mathsf{f}:\mathsf{T}_1\in\Gamma}{\Gamma\vdash\mathsf{f}:\quad}\quad\dfrac{\mathsf{x}:\mathsf{T}_2\in\Gamma}{\Gamma\vdash\mathsf{x}:\quad}}{\Gamma\vdash\mathsf{f\,x}:\quad\ |\ \{\qquad\qquad\qquad\}}}{\Gamma ::= \{\mathsf{f}:\mathsf{T}_1,\ \mathsf{x}:\mathsf{T}_2\}\vdash\mathsf{f(f\,x)}:\quad\ |\ \{\qquad\qquad\qquad\qquad\qquad\qquad\}}}{\{\mathsf{f}:\mathsf{T}_1\}\vdash\lambda\mathsf{x}:\mathsf{T}_2\,.\ \mathsf{f(f\,x)}:\quad\ |\ \{\qquad\qquad\qquad\qquad\qquad\}}}{\{\}\vdash\lambda\,\mathsf{f}:\mathsf{T}_1\,.\ \lambda\mathsf{x}:\mathsf{T}_2\,.\ \mathsf{f(f\,x)}:\quad\ |\ \{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\}}$$

Second pass

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{f}:\mathsf{T}_1\in\Gamma}{\Gamma\vdash\mathsf{f}:\mathsf{T}_1}\qquad\dfrac{\dfrac{\mathsf{f}:\mathsf{T}_1\in\Gamma}{\Gamma\vdash\mathsf{f}:\mathsf{T}_1}\quad\dfrac{\mathsf{x}:\mathsf{T}_2\in\Gamma}{\Gamma\vdash\mathsf{x}:\mathsf{T}_2}}{\Gamma\vdash\mathsf{f\,x}:\mathsf{T}_3\ |\ \{\mathsf{T}_1=\mathsf{T}_2\to\mathsf{T}_3\}}}{\Gamma ::= \{\mathsf{f}:\mathsf{T}_1,\ \mathsf{x}:\mathsf{T}_2\}\vdash\mathsf{f(f\,x)}:\mathsf{T}_4\ |\ C ::= \{\mathsf{T}_1=\mathsf{T}_2\to\mathsf{T}_3,\mathsf{T}_1=\mathsf{T}_3\to\mathsf{T}_4\}}}{\{\mathsf{f}:\mathsf{T}_1\}\vdash\lambda\mathsf{x}:\mathsf{T}_2\,.\ \mathsf{f(f\,x)}:\mathsf{T}_5\ |\ C\cup\{\mathsf{T}_5=\mathsf{T}_2\to\mathsf{T}_4\}}}{\{\}\vdash\lambda\,\mathsf{f}:\mathsf{T}_1\,.\ \lambda\mathsf{x}:\mathsf{T}_2\,.\ \mathsf{f(f\,x)}:\mathsf{T}_6\ |\ C\cup\{\mathsf{T}_5=\mathsf{T}_2\to\mathsf{T}_4,\mathsf{T}_6=\mathsf{T}_1\to\mathsf{T}_5\}}$$

In general, these trees are computed algorithmically. Solving the constraints yields here :

$$\mathsf{T}_2=\mathsf{T}_3=\mathsf{T}_4,\qquad \mathsf{T}_1=\mathsf{T}_4\to\mathsf{T}_4,\qquad \mathsf{T}_5=\mathsf{T}_2\to\mathsf{T}_4,\qquad \mathsf{T}_6=\mathsf{T}_1\to\mathsf{T}_2\to\mathsf{T}_4$$

The following figure summarizes the typing rules augmented with constraints that describe the relations between the type variables :

$$[\text{CT-Ax}]\ \frac{x : \mathsf{T}_1 \in \Gamma}{\Gamma \vdash x : \mathsf{T}_1 \mid \{\}}$$

$$[\text{CT-If}]\ \frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \mid C_1 \qquad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}_2 \mid C_2 \qquad \Gamma \vdash \mathsf{t}_3 : \mathsf{T}_3 \mid C_3}{\Gamma \vdash \left(\mathtt{if}\, \mathsf{t}_1\, \mathtt{then}\, \mathsf{t}_2\, \mathtt{else}\, \mathsf{t}_3\right) : \mathsf{T}_2 \mid C' = C_1 \cup C_2 \cup C_3 \cup \{\mathsf{T}_1 = \mathsf{Bool}\} \cup \{\mathsf{T}_2 = \mathsf{T}_3\}}$$

$$[\text{CT-Abs}]\ \frac{\Gamma, \mathsf{x} : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 \mid C_1}{\Gamma \vdash \left(\lambda \mathsf{x} : \mathsf{T}_1\ .\ \mathsf{t}_2\right) : \mathsf{T}_3 \mid C' = C_1 \cup \{\mathsf{T}_3 = \mathsf{T}_1 \to \mathsf{T}_2\}}$$

$$[\text{CT-App}]\ \frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \mid C_1 \qquad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}_2 \mid C_2}{\Gamma \vdash \left(\mathsf{t}_1\ \mathsf{t}_2\right) : \mathsf{T}_3 \mid C' = C_1 \cup C_2 \cup \{\mathsf{T}_1 = \mathsf{T}_2 \to \mathsf{T}_3\}}$$

4. Explain the constraints for each rule. In particular, comment on the fact that an if-then expression without the "else" part is in general ill-typed.
5. Write the set of constraints for $E \equiv \lambda$ `x:X.` λ `y:Y.` λ `z:Z. ((x) z) ((y) z) : S` using a derivation tree.

Further extending the language may be more or less diffcult depending on the expressive power of the construction that we want to add. We give here two examples :

6. Extend the language with a rule testing the equality of two expressions.
7. Extend the language with a rule allowing recursion.
   What kind of problems does it stir up ?