Bordeaux University
Year 2014-2015

Master ESSV
MtoCPL

TD n°1 - Type systems and derivations

Exercice 1: Basic evaluation

The following example, taken from Pierce's "*Types and Programming Languages*" defines a very simple language on booleans and integers, and its evaluation rules :

Syntax :

`t ::=`		*terms*
	`true`	*constant true*
	`false`	*constant false*
	`if t then t else t`	*conditional*
`v ::=`		*values*
	`true`	*true value*
	`false`	*false value*

Evaluation rules :

$$\texttt{if}\,\texttt{true}\,\texttt{then}\,\texttt{t}_2\,\texttt{else}\,\texttt{t}_3 \rightarrow \texttt{t}_2$$

$$\texttt{if}\,\texttt{false}\,\texttt{then}\,\texttt{t}_2\,\texttt{else}\,\texttt{t}_3 \rightarrow \texttt{t}_3$$

$$\frac{\texttt{t}_1 \rightarrow \texttt{t}_1'}{\texttt{if}\,\texttt{t}_1\,\texttt{then}\,\texttt{t}_2\,\texttt{else}\,\texttt{t}_3 \rightarrow \texttt{if}\,\texttt{t}_1'\,\texttt{then}\,\texttt{t}_2\,\texttt{else}\,\texttt{t}_3}$$

Syntax :

`t ::=`	`...`	*terms*
	`0`	*constant zero*
	`succ t`	*successor*
	`iszero t`	*zero test*
`v ::=`	`...`	*values*
	`natv`	*numeric values*
`natv ::=`		*numeric values*
	`0`	*zero value*
	`succ natv`	*successor value*

Evaluation rules :

$$\frac{\texttt{t}_1 \rightarrow \texttt{t}_1'}{\texttt{succ}\,\texttt{t}_1 \rightarrow \texttt{succ}\,\texttt{t}_1'}$$

$$\texttt{iszero}\,0 \rightarrow \texttt{true}$$

$$\texttt{iszero}(\texttt{succ}\,\texttt{natv}_1) \rightarrow \texttt{false}$$

$$\frac{\texttt{t}_1 \rightarrow \texttt{t}_1'}{\texttt{iszero}\,\texttt{t}_1 \rightarrow \texttt{iszero}\,\texttt{t}_1'}$$

In this diagram, the column on the left represents languages (described by a grammar, rules being added gradually), whereas the right column represents deduction rules. By convention, every symbol $\texttt{t}_{[i]}^{[']}$ represents an element of the language generated by the non-terminal `t`.

1. Give an example of a *stuck* expression in this language. Is the language type-safe ?
2. Explain why there is a distinction between `v` and `natv`.

For the sake of simplicity, the `pred` function has been removed from the language. This function has the particularity to be a partial function on natural numbers.

3. What evaluation rule could we propose for `pred 0` ? What about other values ?

4. Write the evaluation tree of the following expression :

   `if (iszero (pred (succ zero))) then succ 0 else false`

> ▷ OCaml (`http://caml.inria.fr`) is a functional programming language developed at INRIA, distributed with a compiler `ocamlc` and an interaction loop `ocaml`.
> In order to write OCaml code, the most direct way consists in launching `emacs` on a `.ml` file, and then running the `tuareg-mode`. Then, it becomes possible to execute every expression in the interaction loop using the `C-x C-e` shortcut.

Exercice 2: Syntax derivations

Let us represent the code in OCaml using sum types, in the following way :

```
type id   = string                  (* Identifiers *)

type term =
  | TmTrue |  TmFalse               (* Booleans    *)
  | TmZero |  TmSucc of term        (* Naturals    *)
  | TmIsZ  of term                  (* Zero test   *)
  | TmIf   of term * term * term    (* Conditional *)
```

Download and compile the source code given with the tutorials. In order to test this code, it is necessary to start OCaml with the command `ocaml syntax.cma`. Then it becomes possible to parse expressions in the following manner :

```
Parser.parse_term "if true then false else true";;
```

The syntax is the natural one, with the possibilities to add parentheses.

1. Construct the previous term in the OCaml interpreter.

The `bussproofs` LaTeX package allows to build derivation trees in a simple manner :

```
\begin{prooftree}
\AxiomC{}
\UnaryInfC{0}\UnaryInfC{iszero 0}
\AxiomC{}
\UnaryInfC{true}
\AxiomC{}
\UnaryInfC{false}
\TrinaryInfC{if (iszero 0) then t
                          else f}
\end{prooftree}
```

$$\dfrac{\dfrac{\dfrac{}{0}}{\text{iszero } 0} \qquad \dfrac{}{\text{true}} \qquad \dfrac{}{\text{false}}}{\text{if (iszero } 0\text{) then t else f}}$$

2. Construct a function that builds a derivation tree for a term.

You may take inspiration from the `term_to_string` function in the `syntax.ml` file. The code is already designed to print the result string on the standard output when using the `make latex` command, and generate a `syntax.pdf` file.

Exercice 3: Basic types

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

Syntax :

T ::=		*types*
	Bool	*type of booleans*

Typing rules :

$$\texttt{true} : \mathsf{Bool}$$
$$\texttt{false} : \mathsf{Bool}$$

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

Syntax :

T ::=	...	*types*
	Nat	*type of naturals*

Typing rules :

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

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

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

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.