HW1 - Bidirectional Type Checking
Deadline: 16 February 2018

February 3, 2019

This assignment has the following aims:

- To get acquainted with OCaml
- To familiarize yourself with reading rules describing derivable judgements (eg. evaluation judgements and typing judgements) and turning them into code
- To familiarize yourself with the Bindlib library for representing syntax involving binders.

This assignment requires that you install the Bindlib, Menhir, and oUnit libraries.

```
opam update && opam install menhir bindib ounit
```

The structure of this document is as follows. We first introduce the syntax of the language we work with ($\lambda^{\mathbb{B},\to}$). Then we provide the typing rules as seen in lecture, and formulate the submission itself. The document on Canvas introducing Bindlib can be considered a prerequisite.

1 Syntax

The concrete syntax of $\lambda^{\mathbb{B},\to}$ where x represents an identifier, is:

$$
\begin{array}{rcl}
M, N, P, Q & ::= & x \\
& | & \texttt{true} \\
& | & \texttt{false} \\
& | & (M : \sigma) \\
& | & \texttt{if } M \texttt{ then } P \texttt{ else } Q \\
& | & \texttt{lam } x\texttt{. } M \\
& | & (M\ N) \\
\\
\sigma, \tau & ::= & \texttt{bool} \\
& | & \sigma \to \tau
\end{array}
$$

We encode this in OCaml using the following datatypes,

```
type texpr =
  | BoolType
  | FuncType of texpr * texpr

type term =
  | CstTrue
  | CstFalse
  | Var of term Bindlib.var
  | App of term * term
  | Abs of (term, term) Bindlib.binder
  | ITE of term * term * term
  | TDecl of term * texpr
```

The type `texpr` represents the types in our language and `term` the type of terms. Note that the constructor `TDecl` represents terms annotated with type declarations ($M : \sigma$). This annotation is only useful for type-checking purposes; when evaluating it to its normal form we simply ignore, or erase, the type.

As an example, the identity function on booleans $\lambda x.x : \mathbb{B} \to \mathbb{B}$ in the lambda calculus can be represented in our concrete syntax as `(lam x. x : bool -> bool)`. Likewise, the term $(\lambda f.\lambda x.f\ f\ x : (\mathbb{B} \to \mathbb{B}) \to \mathbb{B} \to \mathbb{B})\ (\lambda x.x)$ is represented in our concrete syntax as

```
((lam f. (lam x. (f (f x))): (bool -> bool)-> bool -> bool)(lam x. x))
```

In particular, note that we parenthesize every application and type declaraction.

2 Typing rules

Here is a summary of the typing-rules for $\lambda^{\mathbb{B},\to}$.

$$\frac{x : \sigma \in \Gamma}{\Gamma \rhd x \Rightarrow \sigma}\ (\text{BT-Var})$$

$$\frac{\Gamma, x : \sigma \rhd M \Leftarrow \tau}{\Gamma \rhd \texttt{lam}\ x.\ M \Leftarrow \sigma \to \tau}\ (\text{BT-Abs}) \qquad \frac{\Gamma \rhd M \Rightarrow \sigma \to \tau \quad \Gamma \rhd N \Leftarrow \sigma}{\Gamma \rhd (M\ N) \Rightarrow \tau}\ (\text{BT-App})$$

$$\frac{}{\Gamma \rhd \texttt{true} \Rightarrow \texttt{bool}}\ (\text{BT-True}) \qquad \frac{}{\Gamma \rhd \texttt{false} \Rightarrow \texttt{bool}}\ (\text{BT-False})$$

$$\frac{\Gamma \rhd M \Leftarrow \texttt{bool} \quad \Gamma \rhd P \Leftarrow \sigma \quad \Gamma \rhd Q \Leftarrow \sigma}{\Gamma \rhd \texttt{if}\ M\ \texttt{then}\ P\ \texttt{else}\ Q \Leftarrow \sigma}\ (\text{BT-ITE})$$

$$\frac{\Gamma \rhd M \Leftarrow \sigma}{\Gamma \rhd (M : \sigma) \Rightarrow \sigma}\ (\text{BT-TDecl}) \qquad \frac{\Gamma \rhd M \Rightarrow \tau \quad \tau = \sigma}{\Gamma \rhd M \Leftarrow \sigma}\ (\text{BT-ChkInf})$$

3 Your task

Implement a bidirectional type-checker for $\lambda^{\mathbb{B},\to}$, using the rules given in the section above as a specification. You will be provided with a stub which includes the following modules.

- (`ast.ml`) AST. The following functions form the interface. We also define an additional ADT for terms not shown here. The source contains details about why this is necessary.

```
type term         (* terms *)
type texpr        (* types *)

val fv : term -> string list

val lift_term : term -> term Bindlib.box

val lift_type : texpr -> texpr Bindlib.box

val nf : term -> term

val string_of_term : term -> string

val string_of_type : texpr -> string
```

- (`tc.ml`) Type-checker (to implement). The type `'a error` is the return type of your type-checking function. An appropriate error message indicating why type-checking might have failed should be provided. Keep in mind that `synthesize` and `check` are defined as two mutually recursive functions.

  The type `texpr Ctx.t` represents maps from strings to type expressions in $\lambda^{\mathbb{B},\to}$. While the stub uses `Map.Make`, you are free to use any other representation for type environments. However, make sure that your top level type-checking function `tc` calls `synthesize` with the appropriate value for an empty environment.

  Note: it is also appropriate to have `check` return `unit error` instead of `texpr error`. The latter is chosen for simplicity. While `bool` might be another possibility, it complicates the process of producing error messages.

```
type 'a error = OK of 'a | Error of string

module Ctx = Map.Make(String)

val synthesize : texpr Ctx.t -> term -> texpr error
val check : texpr Ctx.t -> term -> texpr -> texpr error

val tc : texpr -> texpr error
```

- (`lexer.mll, parser.mly`) A simple lexer and parser for $\lambda^{\mathbb{B},\to}$.
- (`test.ml`) Contains functions to help parse terms and a collection of tests. This module requires oUnit. Note that the test cases are meant to be type checked in an empty environment. For parsing, the following two functions are given g

```
val parse_term : string -> Ast.term

val parse_type : string -> Ast.texpr
```

4 Compiling and Testing

You will be able to compile everything required for this assignment using the `Makefile` provided in the stub. Run `make` to compile the type checker and run `make test` to run the provided

test suite.

In order to test your code using utop, you can run `make utop`. Make sure to additionally run `#load_rec "tc.cmo"` and/or `#load_rec "test.cmo"` to load the appropriate modules. The test module should be loaded in order to use the parser. Once this is done, you should be able to run

```
Tc.tc @@ Test.parse_term "(lam x. x : bool -> bool)";;
```

and other terms in utop to test your type checker. Since the `test` module exists if any of the tests fail, you may have to temporariliy disable running the tests while you develop your program.

5 Your Submission

Hand in a zip file with all your sources on Canvas. Make sure you have `#use "topfind";;` in your `.ocamlinit` file (which adds a new directive named `#require` that you can use to load your opam packages into the toplevel) and then `#require "bindlib";;`