HW2 – Syntax and Type Checking

CS 476, Fall 2018
Due Sep. 19

1 Instructions

Begin by downloading the file `hw2-base.ml` from the course website, and renaming it to `hw2.ml`. This file contains the outline of the homework, with incomplete definitions commented out. For each problem, remove the comments and fill in the associated definitions. You can also add functions and modify the outline as you see fit. Make sure to answer the problems in both Part 2 (BNF Grammars and ASTs) and Part 3 (Type Checking). Submit your completed `hw2.ml` via Gradescope. As always, please don't hesitate to ask for help on Piazza (`https://piazza.com/class/jkh8q52qrh06v`).

2 BNF Grammars and ASTs

Here is a BNF grammar:

$$A ::= \langle \mathit{ident} \rangle \mid A \texttt{ on } A \mid \texttt{fun } \langle \mathit{ident} \rangle \texttt{ -> } A$$
$$D ::= \texttt{let } \langle \mathit{ident} \rangle \texttt{ = } A \mid \texttt{let rec } \langle \mathit{ident} \rangle \texttt{ = } A$$

1. (4 points) Write datatypes `ast_A` and `ast_D` of abstract syntax trees for A and D respectively, where $\langle \mathit{ident} \rangle$ is represented by the `string` type. This grammar is completely separate from the grammar for expressions, so make sure that your constructors *do not* include arguments of type `exp`.

2. (2 points) Define a value `tree1 : ast_D` that is an AST for the term

   ```
   let y = fun f -> (fun x -> f on x) on f
   ```

3. (3 points) Write a function `count_funs : ast_D -> int` that counts the number of `fun` nodes in an AST for D. As an example, `count_funs tree1` should return 2.

   Hint: start by writing a function `count_funs_A` that counts the number of `fun` nodes in an AST for A.

4. (3 points) Write a function `count_id : string -> ast_D -> int` that takes a string `s` and an AST for D and counts the number of times `s` appears as an identifier in the AST. As an example, `count_id "f" tree1` should return 3.

   Hint: as above, start by writing a function that counts identifiers for A.

3 Type Checking

The rules of the type system for expressions are:

$$\frac{n \text{ is a number}}{n : \text{int}} \qquad\qquad \frac{n \text{ is a boolean}}{n : \text{bool}}$$

$$\frac{e_1 : \text{int} \qquad e_2 : \text{int}}{e_1 \oplus e_2 : \text{int}} \text{ where } \oplus \text{ is an arithmetic operator}$$

$$\frac{e_1 : \text{bool} \qquad e_2 : \text{bool}}{e_1 \otimes e_2 : \text{bool}} \text{ where } \otimes \text{ is a boolean operator}$$

$$\frac{e_1 : \tau \qquad e_2 : \tau}{e_1 = e_2 : \text{bool}} \qquad\qquad \frac{e : \text{bool} \qquad e_1 : \tau \qquad e_2 : \tau}{\text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

The provided `hw2-base.ml` includes a type `exp` of ASTs for expressions, and a type `typ` for types. `Tint` is the type of ints in the expression language, and `Tbool` is the type of bools in the expression language; be careful not to confuse them with `int` and `bool`, which are types in OCaml itself.

(13 points) Write a function `typecheck : exp -> typ -> bool` that returns `true` if the given expression has the given type, and `false` otherwise. For instance, `typecheck (Add (Int 3, Int 4)) Tint` should be `true`, and `typecheck (If (Bool false, Int 3, Bool true)) Tbool` should be `false`.

Hint: the structure of the function will closely resemble that of the interpreter `eval` that we built in class, and the typing rules tell you what to do in each case. In the `Eq` case, you will have to try both possible values of τ, `Tint` and `Tbool`.