Assignment 2: Typed Language Implementation

Due: Feb 11 2016 1AM
Support code:https://www.cs.umass.edu/~arjun/courses/cmpsci631-spring2016//hw/interp.zip

The high-level goal of this assignment is to implement an interpreter and type-checker for a small programming language. In addition, you need to hand in a PDF with a formal semantics and type system for lists, which are part of the language.

1 Restrictions

You can use any features of OCaml you like. We strongly recommend avoiding objects. Imperative features aren't necessary, but may be helpful for implementing recursion.

2 Support Code

The support code has the following files:

- `Syntax.ml` defines the abstract syntax of the language. Almost all features have been covered in class (and in lecture notes). However, the language has two new features, which are documented below. You can add cases to the language if you like, but do not change the existing cases.
- `Parser.mly` and `Lexer.mll` define a parser and a lexer for the language. Feel free to change them if you like, but you can use them as-is.
- `ParserHelpers.ml` defines two functions, `from_file` and `from_string` that you can use to parse expressions. You should use these functions instead of using the parser directly.
- `Main.ml` is where you should define the main type-checking and evaluation functions.

3 Lists

The language in this assignment includes lists. The follow examples show how lists are evaluated:

```
TEST "a list of three integers" =
  ParserHelpers.from_string "1 :: 2 :: 3 :: empty<int>" =
    Cons(Const (Int 1), Cons (Const (Int 2), Cons (Const (Int 3), Empty TInt)))

TEST "an empty list of booleans" =
  ParserHelpers.from_string "empty<bool>" =
    Empty TBool

TEST "testing the head function" =
  eval (ParserHelpers.from_string "head (1 :: empty<int>)") =
  eval (ParserHelpers.from_string "1")

TEST "testing the tail function" =
  eval (ParserHelpers.from_string "tail (1 :: 2 :: empty<int>)") =
  eval (ParserHelpers.from_string "2 :: empty<int>")

TEST "testing the empty? function (false)" =
  eval (ParserHelpers.from_string "empty? (1 :: 2 :: empty<int>)") =
  eval (ParserHelpers.from_string "false")

TEST "testing the empty? function (true)" =
  eval (ParserHelpers.from_string "empty? empty<int>") =
  eval (ParserHelpers.from_string "true")
```

You need to formalize the semantics and typing rules for lists and implement them. Submit the semantics and typing rules as a PDF.

4 Recursion

The language includes a construct $\mathsf{fix}\ x : T.e$ that you can use to implement recursive functions. The following rule describes its semantics:

$$\textsc{Fix}\ \frac{e[x \mapsto \mathsf{fix}\ x : T.e] \Downarrow v}{\mathsf{fix}\ x : T.e \Downarrow v}$$

The intuition behind this rule is that we evaluate the body e by substituting x with the original expression. Therefore, e can refer to itself via x.

The following rule describes how it should be type-checked:

$$\textsc{T-Fix}\ \frac{\Gamma[x \mapsto T] \vdash e : T}{\Gamma \vdash \mathsf{fix}\ x : T.e : T}$$

Typically, the type T will be a function type (thus, e will be a function).

5 Handin

Handin a ZIP/TGZ file on Moodle with:

- A complete type checker and evaluator,
- Test cases that show you've implemented recursion correctly (e.g., just the factorial function is sufficient), and
- A PDF with the semantics and typing rules for lists.