CSC 200(H) The `TyExprC` Type Checker

January 31, 2020

1 Introduction

Chapter 27 of the PAPL book defines a typed language with minimal integer, Boolean and function operations. We call the language `TyExprC`. In this assignment, you will either work alone or collaborate with a partner to implement the type checker for `TyExprC`. You will program the type checker in Rust.

Type checking is the process of proving type correctness. The type checker is an automated tool to prove (a weaker condition of) program correctness. Given a `TyExprC` program, the `TyExprC` type checker either proves that the program is type correct or it rejects the program.

The language `TyExprC` represents the essence of functional programming. As a language, it is slightly larger than lambda calculus (so easier to program in) but still small enough (so a student can implement a complete type checker in a class project).

2 The `TyExprC` Language

The `TyExprC` language has the following syntax:

```
TyExprC :=
    numC(Num)
  | plusC(TyExprC, TyExprC)
  | multC(TyExprC, TyExprC)
  | trueC
  | falseC
  | eqC(TyExprC, TyExprC)
  | ifC(TyExprC, TyExprC, TyExprC)
  | idC(String)
  | appC(TyExprC, TyExprC)
  | fdC(String, Type, Type, TyExprC)
  | recC(String, String, Type, Type, TyExprC, TyExprC)

Type :=
    numT
  | boolT
  | funT(Type, Type)
```

In this grammar, the parentheses and commas are terminal symbols; a terminal string (i.e. a string literal) starts with a non-capital letter, e.g. `numC, trueC`; and a non-terminal begins with a capital letter, i.e. `TyExprC, Num, String, Type`. Of the four non-terminals, `TyExprC, Type` are defined by the grammar. Of the remaining two, `Num` is a non-negative integer, and `String` is a string that is not one of the terminal strings in the grammar. Check the textbook *Programming Language Pragmatics* if you need a (formal) regular-language definition for numbers and strings.

Any `TyExprC` expression is a `TyExprC` program. For example, `trueC` is a program that evaluates to true. Read Chapter 27 to see how the syntax of `TyExprC` is used in programming. For example, `fdC` defines a function with a single parameter. Its four arguments in `fdC(String, Type, Type, TyExprC)` are parameter name, parameter type, return type, and the function body respectively. A function defined by `fdC` is unnamed, a.k.a. an anonymous function or a lambda. In comparison, `recC` defines a named function, so the function can be recursive.

Recall that a program has names, values and constants. In this language, the only way to introduce a name is by defining it in one of the two function definitions: by `fdC` which defines a parameter or by `recC` which defines a function name and a parameter. When either type of names is used, e.g. a parameter in `plusC` or a function in `appC`, it must use `idC(String)` to "access" the parameter or the function by its name.

For example, the following program defines an identity function on a number:

```
fdC("n", numT, numT, idC("n"))
```

We will go over a recursive factorial function in-class.

3 Collaboration

Partnering You may choose to complete the assignment by yourself or with a partner. If you work alone, your type checker must support the full language except for the recursive function, i.e. `recC`, which is (5%) extra credit. If you work in a pair, you must support the full language including the recursive function.

When choosing a partner, we ask you to avoid choosing your best friend or someone you know very well. Also we ask you not to choose someone you do not know. Ideally, choose someone you know but do not know well.

Code Sharing We encourage students to collaborate with people who are not your partner. You are always allowed to discuss and share ideas. In addition, we allow students to share the scanner and parser code of the type checker. Your type checker can use the scanner and/or parser implementation of another student, as long as you acknowledge the source of the code. It is still your responsibility that your type checker works correctly. Any problem with the code used by your type checker counts as your problem.

You must not share your code for type checking or use others' code for type checking.

4 Requirements

You will implement the `TyExprC` type checker called `tc200` in Rust.

1. The checker program `tc200` takes an input file name. It reads the content of the file as a `TyExprC` program, performs the type check, and returns either the result type of the program or an error message saying that the program fails the type check.
2. As unit tests, provide a test program for each of the clauses given in the `TyExprC` grammar.

5 Due Date and Submission

Your final submission should be a compressed folder, uploaded to Blackboard, containing your project directory. ***Please name your compressed folder: prog_assign3_netid***

The due date for this assignment is: **Sunday, February 9th, 2020 at 11:59 PM.**