# COMP 325 - Interpreter 5 - Static Types

Fall 2015

**Abstract**

Your final interpreter is a very basic extension of the simple typed language discussed in the early parts of chapter 16 of the text. In doing so you get to explore how type checkers and interpreters work together to produce safe (see chapter 17) languages. The emphasis of this assignment will be a robust, well documented set of tests as well as the core type checker and interpreter. There is no desugar and no parse.

## Expressions

In addition to all the expressions in *TyExprC* discussed in 16.1-16.5 of the text, your language must provide typed versions the following expressions.

- A binary division operator
- A basic, single binding *let* expression (Interpret this directly. Do not desugar.)
- Binary > and == operators
- *n*-ary functions (as opposed to the unary functions discussed in the book). Note: This adds a new dimension to function types.

These expressions should be typed checked by the function *tc* and interpreted by the *interp* function.

## Type Checker Tests

The tests for your type checker (tc) are to be extremely well done and documented. Rather than place them all in a *where* block with the *tc* definition, they should be broken down into logical groups and placed in appropriately named *check* blocks. Tests within each block should be documented with comments to the point that reading tests and comments provides a detailed understanding of the type checker's behavior and capabilities. Included with comments should be syntactic representations of the program being tested so that readers see the code as the programmer may have written it rather than just the abstract representation used by the type checker and interpreter. It is OK if you comment on several related tests with a single comment block, just be sure you cover all your bases when you do so.

## Interp

You've interpreted all of the expressions in this language before with the exception of a primitive *let*, which was previously desugared before interpretation. Your goal for this interpreter is not to explore new semantics, but to reduce the code to the minimal amount necessary by removing all error checking that is carried out by the type-checker leaving only the errors that must be caught dynamically at run-time.

## Big Picture Tests: Safe and Sound

Your small language should be *safe*. To demonstrate this, design and run a series of tests that clearly illustrate that no meaningless computation can occur in this language. These tests should utilized the combined effort of the type checker followed by the interpreter. Break the tests into logical groups and put each group in a well-named *check* block. As before, document the tests with comments and give a syntactic representation of the code being tested. In your documentation, be sure to clearly state what part of the system (*tc* or *interp*) detects and catches the error.

## Logistics

For this assignment you only need to write the expression type checker and interpreter, and any necessary helpers for those functions. No parser, no top-level definition handlers, no user-interface (i.e. run) function, and no desugarer. The completed assignment is due on **12/8**. The grade is determined as follows:

| Area | Points |
|---|---|
| tc tests | 20 |
| tc | 20 |
| Big picture tests | 15 |
| interp | 10 |
| Data Definitions | 5 |
| Style and Comments | 5 |
| | 75 total |

You are expected to have all required expressions represented in all parts of your design. If it's not there and at least stubbed, then you lose points. At this point we should be able to lay out the skeleton of top level cases. Sufficiency of testing is covered for the section your testing. The style and comment part of your grade accounts for good coding practices like proper indentation, avoiding printed line wrapping, good identifier and function names, documentation, and commenting of logic.