CS 6620
assigned: Jan. 21, 2020
12 points **Homework 1** due: $\leq$23:59 EST Feb. 04, 2020

This assignment should be completed individually.

Part 1

Complete the TIP type analysis by: a implementing the undefined behavior in `tip.analysis.TypeAnalysis.visit`, explaining three distinct cases of the type analysis that implemented , and providing evidence that your implementation is correct. More specifically, you should submit the following files to the CS6620 collab site for HW1.

1. `TypeAnalysis.scala` with all `// <--- Complete here` text replaced with functional scala code.
2. A PDF file `hw1.pdf` with the following information for part 1:
   (a) Describe three distinct cases of type analysis that you implemented; "distinct" means that the structure of the generated type constraints are not simple variants of one another, e.g., renamings of parameters.
   (b) Provide evidence of the correctness of your type implementation. For example, you might describe how you tested your implementation and why you think it is adequate.
   (c) Explain how the `TypeAnalysis` and `UnionFindSolver` operate when processing the following simple TIP program: `poly(p) { return *p; }`

Part 2

Solve Exercise 3.14 in the SPA notes. This exercise asks you to extend the type analysis in TIP to support arrays. Your solution should focus on the *design* of the extension, e.g., what are the type rules, and a description of the types inferred for the example – you will conduct this type analysis by hand.

Add to the PDF file `hw1.pdf` the following information for part 2:

1. Describe the type rules for all of the array operations.
2. For the given example, provide the inferred types for the 4 program variables.

Hints

You will want to become familiar with the following elements of the TIP analysis implementation:

- `tip.ast.Ast` and `tip.types.Types`
- the `TypeAnalysis` makes use of `tip.ast.DepthFirstAstVisitor` not once, but twice; to generate and solve constraints and then to collect up the inferred types for each identifier and expression
- `tip.ast.DepthFirstAstVisitor` is an instantiation of the visitor pattern for the `tip.ast.Ast`; this is a powerful design pattern that appears frequently in program analyzers
- the `map` function is a clean way of transforming a list. A call to $l.map(f)$ will produce a new list by applying f to each element of l, i.e., if l_i is the ith element of l, then $f(l_i)$ is the ith element of $l.map(f)$.
- the main task in this assignment is to figure out how to map AST nodes to constraints that either unify terms for an AST node with a type or unify terms for two AST nodes.