

## Review Set 7

This Review Set asks you to prepare written answers to questions on garbage collection and exceptions. Each of the questions has a short answer. You may discuss this Review Set with other students and work on the problems together. However, your write-up should be your own individual work.

1. Consider *Stop & Copy* vs. *Mark & Sweep* garbage collection.
  - (a) Assume that the garbage collector is run only when the user program runs out of memory, i.e. when a call to `new` cannot be satisfied. Is one of these two GC algorithms ‘faster’ than the other? Which algorithm needs to be run more frequently?
  - (b) Does either algorithm use strictly more memory than the other?

Python uses reference counting for its garbage collector. It uses a special ‘cycle detector’ to clean up cyclical data structures periodically.

- (c) Are reference cycles common in everyday data structures?
  - (d) Briefly describe how one might implement a cycle detector. When can a cycle be cleaned?
2. Imagine that we have an updated version of Cool that supports `try`, `catch` and `throw` (as in the ‘Exceptions and Error Handling’ lecture notes). Suppose further that we want to add a new construct to the language: `protect e`, which works as follows:

- At compile time, the type checker verifies that the expression  $e$  does not have any uncaught exceptions. In other words,  $e$  can contain `throw` clauses, but they must all be caught using a `try / catch` construct. If  $e$  does not meet this requirement, the compiler should reject the program.
- At runtime, `protect e` simply evaluates to  $e$ . Another way to look at this is that the `protect` keyword is ignored at runtime.

In order to check `protect` expressions at compile time, we need to extend our typing judgments to track a boolean  $E$  indicating whether an exception can be thrown out of the expression:

$$O, M, C \vdash e : T, E$$

For example, the extended rule for `+` would be:

$$\frac{\begin{array}{c} O, M, C \vdash e_1 : Int, E_1 \\ O, M, C \vdash e_2 : Int, E_2 \end{array}}{O, M, C \vdash e_1 + e_2 : Int, E_1 \vee E_2} \quad \text{[Plus]}$$

- (a) Assume that a `try/catch` block catches all exceptions (i.e. the exception type being caught is always `Object`).

Give the new type rules for `protect`, `try/catch`, and `throw`. For example, the rule for `try/catch` should look similar to this:

$$\frac{\dots}{O, M, C \vdash \text{try } e_0 \text{ catch } x : \mathbf{Object} \Rightarrow e_1 : T_0 \sqcup T_1, E'}$$

- (b) The typing system described here is not necessarily safe. Give a code sample that illustrates this. Your code should pass the type checker, but still manage to throw an exception out of a `protect` statement. Briefly explain how your code manages to fool the type checker.