EECS 598 and EECS 498 (Fall 2021) – Assignment 3
Due date: Wednesday, October 6 at 11:59pm EST

Collaboration. You are free to discuss the assignment with others or work together towards the solution. However, all of your code must be written by yourself. Your write-up must be your own as well. **Please list your collaborators at the top of your submissions.**

Goal. The goal of this assignment is to help you understand search prioritization and how it can help speed up program synthesis techniques (in particular, our top-down search implementation).

1 Getting started

Download `A3` from
`https://web.eecs.umich.edu/~xwangsd/courses/f21/assignments/pa3.zip`
which contains a `pa3` folder. Put it under `syn`.

```
codebase
└─src
  └─main
    └─java
      └─syn
        ├─base
        ├─pa0
        ├─pa1
        ├─pa2
        └─pa3
          ├─Synthesizer3.java
          ├─Test8.java
          └─Test9.java
```

2 Prioritize search

In addition to pruning, prioritization is another important idea to speed up program synthesis. Intuitively, search prioritization means *in what order programs should be searched*, whereas pruning is concerned with *what programs are guaranteed to be incorrect*. These two ideas are not completely independent sometimes. For example, one can view pruning as a *hard* version of prioritization: programs that are pruned away will be explored at the very end of the search process (which basically means they will never be explored since it is almost impossible to search exhaustively in practice). Similarly, prioritization can be viewed as a *soft* form of pruning: programs with low priority are "pruned away". However, we cannot really eliminate them away since those programs may not be provably incorrect, but given that we do not have infinite amount of time, it is sometimes okay in practice.

In this assignment, you will implement two search prioritization strategies in `Synthesizer3`. Specifically, the `run` procedure in `Synthesizer3` is modified to implement *worklist* as a priority queue. In particular, each AST (both partial and complete) is assigned a *cost*: a program with a higher cost will have a lower priority and will be put towards the end of the queue. **You will implement the `computeAstCost`**

function that computes the cost for any AST. You can define whatever heuristics here as you want. Some ideas that you may consider:

- ASTs with duplicate R operators have higher costs. For example, you may not want to first search programs like `unite(unite(x,?,?,?),?,?,?)`, although it's possible these programs also satisfy the user-provided example.
- ASTs with duplicate arguments get higher costs. For instance, it may not make much sense to first consider programs such as `unite(x,?,1,1)` or `gather(x,tmp1,tmp2,2,3)`.

Note that we disabled pruning in `Synthesizer3` by making `attemptToPrune` always return false. This is to separate search prioritization from pruning in order to understand how much each of them helps speed up the search. However, in practice, one always combines both to achieve the best performance.

3 Test `Synthesizer3`

You will test your `Synthesizer3` on `Test8` and `Test9`. Please observe the synthesizer's behavior, take screenshots of its outputs, and try to understand and explain the results. Note that both tests extend previous tests from Assignment 2.

4 Submission

Your Task. There are four tasks in this assignment.

- (60 points) Task 1: Implement `Synthesizer3` by completing the `compareAstCost` function.
- (10 points) Task 2: Run `Test8` and take a screenshot of its output.
- (10 points) Task 3: Run `Test9` and take a screenshot of its output.
- (20 points) Task 4: Explain what you observed in `Test8` and `Test9`.
- (30 points, bonus) Task 5: Run your `Synthesizer2` and `Synthesizer3` using at least 3 new test cases. Compare their performance and explain what you observed.

Canvas. Submit the following in one single compressed file (e.g., in .zip):

- **Your code** for Task 1, including **only** `Synthesizer3.java`.
- **Your report** in PDF that includes your screenshots and your explanations.

References