University of Alberta
Computing Science Department
Program Synthesis in XAI - W21

Assignment 1
20 Marks
Due date: 05/02 at 23:59

# Overview

In this assignment you will implement the enumerative algorithms bottom-up search (BUS) and breadth-first search (BFS) we discussed in class. It is fine to discuss the assignment with your classmates, but do not share your code with them. We will use the forum on eClass for discuss issues related to the assignment.

# Description

I have attached a Python file to this assignment with the starter code. The starter code contains an implementation of the abstract syntax tree (AST) for the language you will use to evaluate BUS and BFS, and an implementation of an interpreter for the DSL. The starter code also includes a few test cases.

We will use the following DSL, where $x, y, \cdots$ are variables provided as input and $1, 2, \cdots$ are constant values that will depend on each test case.

$$S \to x \mid y \mid \cdots \mid 1 \mid 2 \mid \cdots \mid (S + S) \mid (S * S) \mid \text{if } B \text{ then } S \text{ else } S$$
$$B \to (S < S) \mid B \text{ and } B \mid \text{not } B$$

1. **(7 Marks)** Implement BUS. Your implementation should:
   (a) Detect equivalent programs and keep in memory only one program that produces a given set of outputs.
   (b) In class we assumed that the grow function could combine any subset of programs generated in previous iterations of the algorithm to generate the next batch of programs. We will do it differently in our implementation. In each iteration we will generate all programs with the smallest ASTs larger than the ASTs generated in the previous iteration. Here, size is the number of nodes in the AST. In iteration one we fill `plist` with all terminal symbols of the language, whose ASTs have size 1 (a single node). In the next iteration we combine the existing ASTs of size 1 into ASTs with the smallest possible size that is larger than 1. For example, if 5 is the size of the smallest AST larger than 1 that we can generate by combining ASTs of size 1, then in iteration 2 we generate all possible ASTs of size 5, but no ASTs of larger size.
   (c) Implement constraints to ensure that the search will obey the structure of the DSL above. See the list below where I refer to operations implemented in the starter code. These constraints can be implemented in the grow function of each operation.
      i. The operation "less than" (`Lt`) should only accept `Var`, `Num`, `Plus`, and `Times` as replacements for its non-terminal symbols. For example, `x < y` is valid, but `If then else < x` isn't.
      ii. The operation "if-then-else" (`Ite`) should only accept the Boolean nodes `And`, `Not`, and `Lt` as possible replacements for its Boolean non-terminal symbol.
      iii. The operations `And` and `Not` should only accept `Lt` as replacements for its non-terminal symbol. According to the DSL other `And` and `Not` operations should be accepted too, but they won't be necessary.

The goal of adding these restrictions is to reduce the search space. Feel free to add more restrictions if you believe they will further reduce the search space while not making the test cases unsolvable.

2. **(7 Marks)** Implement BFS. You should also implement the same constraints implemented for BUS to restrict the set of programs considered during search.

3. **(6 Marks)** Write a report comparing BUS with BFS in terms of running time, number of programs generated, and number of programs evaluated. Explain the results you obtained, in particular, explain the differences in terms of number of programs evaluated and generated until a solution is encountered. Please upload your report, as a pdf file, and a zip file with your implementation to eClass.