University of Alberta
Computing Science Department
Neuro-Symbolic Programming - W23

Assignment 1
20 Marks
Due date: 10/02 at 23:59

Overview

In this assignment you will implement the enumerative algorithms bottom-up search (BUS) and the top-down representative 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 Discord to discuss issues related to the assignment.

Description

The starter code for the assignment 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.

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

The program space can be fairly large, even for the simple problems we are going to solve in this assignment. That is why in the test cases we specify a subset of DSL that will be used to solve a given problem. The signature of the search algorithms will be as follows:

```
def synthesize(bound, operations, integer_values, variables, input_output)
```

The parameter `bound` specifies the maximum size of programs the search algorithm will evaluate. If the search algorithm doesn't encounter a solution within the bound, then it should return failure. The variables `operations`, `integer_values`, and `variables` specify the operations, integer values, and variables to consider in search—this is where we define a subset of the DSL to be used in search. Finally, `input_output` is a dictionary with the set of input values and desired output values. A solution to a problem should correctly map all inputs to their corresponding output. Here is an example of how we are going to invoke the search.

```
search.synthesize(10, [Lt, Ite], [1, 2], ['x', 'y'],
                      [{'x':5, 'y': 10, 'out':5}, {'x':10, 'y': 5, 'out':5},
                      {'x':4, 'y': 3, 'out':3}])
```

In this search we will only consider the less-than symbol and if-then-else structures; the search shouldn't consider multiplication or `and` operators, for example. The search will consider two integer values: 1 and 2 as well as two variables x and y.

1. **(7 Marks)** Implement BUS. Your implementation should:
   (a) Detect and eliminate observational equivalent programs.

(b) The search should be done by increased AST size. That is, in iteration we fill the bank of programs with all terminal symbols of the language, whose ASTs are of 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. If 5 is the size of the smallest AST larger than 1 that we can generate, 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 obeys the structure of the DSL above. 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` programs as replacement for its non-terminal symbol.

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 without 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 on eClass.

   The report doesn't have to include many experiments. The results for the test cases should be sufficient.