Assignment 1

due 2/27/2015 11:59pm.

The goal of this assignment is for you to experiment with some of the different inductive synthesis approaches discussed in class. All the starter code for this assignment is in a file called Assignment1.js; there is also a file default.html that you can use to run the code.

We have defined a simple language for the following expressions:

```
expr:=    num(int n)                                // numeric constant num:int
        | flse()                                    // constant FALSE. Unfortunately ,we can't call it false
        | vr(string varname)                        // variable   var:int
        | plus(expr left, expr right)               // arithmetic expressions plus and times
        | times(expr left, expr right)              //  plus(int,int):int     times(int, int):int
        | lt(expr left, expr right)                 //less than operator lt(int, int):int
        | and(expr left, expr right)                // boolean operators
        | not(expr left)                            // and(bool,bool):bool          not(bool):bool
        | ite(expr cond, expr tcase, expr fcase)    // if-then-else;  ite(bool, int, int):int
```

As part of the starter code for this project, we have included constructors for all the different AST nodes, as well as a basic interpreter that you can use to evaluate expressions in this language. Note that even though all the AST nodes are of type expr, the language they represent has a not-quite-trivial type system on top of it.

Problem 1: 35 pts

The goal of this problem is to show how assumptions about the space of possible expressions can affect both the quality of solutions and the expected solution time for simple synthesis problems.

1.a) Implement a simple random AST generator. Your generator should have the following interface:

function randomExpr(globalBnd, intOps, boolOps, vars, consts)

globalBnd : A bound on the maximum depth allowed for the generated ASTs.
intOps: A list of the integer AST nodes the generator is allowed to use.
boolOps: A list of the boolean AST nodes the generator is allowed to use.
vars: A list of all the variable names that can appear in the generated expressions.
consts: A list of all the integer constants that can appear in the generated expressions.

Make sure your synthesizer does not unnecessarily generate expressions that violate the types of the underlying language; for example, you should never generate ite(5, 7, false). If the bound is reached, the generator can produce one of the terminal nodes (num, flse, vr) even if it was not in the intOps or boolOps list. You should never produce a num if the consts list is empty or a vr if the vars list is empty.

1.b) Write a function that runs the random generator with the parameters below for 1000 times and returns the number of expressions that satisfy the examples given:

globalBnd = 3, intOps=[PLUS, TIMES, VR], booleanOps=[], vars=["x", "y"], consts=[]

x=5, y=5, out=15;
x=8, y=3, out=14;
x=1234, y=227, out=1688;

Print the set of unique expressions satisfying the examples. How big is the set, does it contain the expressions you expected?

1.c) Repeat the experiment above, but with the following parameters:

globalBnd = 3, intOps=[PLUS, TIMES, VR, NUM], booleanOps=[], vars=["x", "y"], consts=[1,2,3,4]

Discuss what changed compared to the previous experiment. How did the change in the search space affect the probability of finding a correct solution? How did it affect the quality and variety of unique solutions?

1.d) Repeat the experiment above but with the following parameters:

globalBnd = 4, intOps=[PLUS, TIMES, VR, NUM], booleanOps=[], vars=["x", "y"], consts=[1,2,3,4]

Discuss what changed compared to the previous experiment. How did the change in the search space affect the probability of finding a correct solution? How did it affect the quality and variety of unique solutions?

1.e) Use your simple synthesizer to generate a function matching the following examples:

x=10, y=7, out=17
x=4, y=7, out=-7
x=10, y=3, out=13
x=1, y=-7, out=-6
x=1, y=8, out=-8

This function requires the complete language and a not-so-small AST depth. Can your synthesizer produce this function? If so, how many trails does it take on average?

1.f) Create a new version of your random synthesizer that incorporates the following biases:

> Multiplications usually occur only between variables and constants or between two variables
> It is rare to have complicated boolean logic
> We do not expect to see long chains of additions

You need to quantify the above statements in terms of probabilities; for example, you can say that if you are chosing an operator and it is an 'and' then the probability that any of its children are an and is reduced. How do these biases affect the results of experiments 1.c and 1.d?

Problem 2) 30 pts

Consider the following grammar for a problem:

```
term: 2*x + ?? |  x*x + ?? | 3*x + ??


prog: if(x < ??){ return term; }
      else if(x < ??){ return term; }
      else if(x < ??){ return term; }
      else { return term; }
```

The question marks ?? indicate unknown constants. Assume the function only has to operate on the domain x\in [0,100] and that its range is [0,10000]. You can also assume the function operates only over the integers.

Even though the space of potential expression is very large, the search space has a lot of structure. Our goal for this problem is to come up with a representation of the search space that exploits this structure and allows us to efficiently solve for a program in this space given a set of input/output pairs. In particular, the goal is to find a way to factor the search space in a way that allows us to solve for the unknowns without having to search the exponentially large space.

Hints:

- Every input will be processed by exactly one term.
- For a given input, if we know what term it corresponds to, we can easily solve for the unknown in that term.
- It may help to think about question 2.b first.

2.a) Implement an algorithm that solves the problem in linear time in the number of samples by properly factoring the search space. Briefly explain your algorithm. Use pictures to explain the main datastructures if you feel that will help.

2.b) How would your algorithm change if you didn't know that there were exactly 4 partitions and your goal was to find a solution with a minimal number of partitions?

Problem 3) 35 pts

For this problem, you will be using Sketch to answer some of the questions from problem 1. You should download the latest version of sketch from here:

http://people.csail.mit.edu/asolar/sketch-1.6.8.tar.gz

This version is similar to 1.6.7 on my website, but includes a few bug fixes. You will find the sketch language reference useful:

http://people.csail.mit.edu/asolar/manual.pdf

Starter code for this assignment is provided in question3.sk

3.a) Encode the grammar from problem 1 as a generator in Sketch. You will essentially be writing the generator equivalent of the randomExpr function from problem 1, but it does not have to be parameterized by intOps and boolOps (see the starter code).

3.b) Use your generator to synthesize expressions for the examples in questions 1.b and 1.e.

3.c) Write a generator that imposes the restrictions from 1.f as hard constraints on the generator, and do problem 3.b again. How do the results compare?

3.d) Try to push your two generators to produce the largest functions that will synthesize in less than five minutes. What is the largest function you can synthesize with 3.a? With 3.b?

Submission Instructions:

Collect all your code in a tar file named with your MIT user name; i.e. myusername.tar The tar file should also include answers to the questions in either a text file or a PDF file. All pset submissions will be done through stellar. If you have questions, please ask!