Project for Program Analysis (263-2910-00L)

Spring Semester, 2015, ETH Zurich

Contact: veselin.raychev@inf.ethz.ch

Revision A. The document may be updated in case questions arise.

Problem

The goal of this project is to create a tool that synthesizes an "inverse" function of a given function implemented in Python. Each synthesized function must follow a given template with some parts of the function given and other details left to be inferred by the tool.

Consider the following example, which takes two numbers `x` and `y`, and returns a tuple containing their sum plus one, as well as the first number.

```
def f(x, y):
  return x+y+1, x
```

The goal of the project is to create a tool that generates an inverse function:

```
def f_inv(p, q):
  return q, p + (-q) + (-1)
```

The inverse function is generated using a template that looks like:

```
def f_inv(p, q):
  return unknown_choice(p,q), unknown_choice(p,q) +
      unknown_choice(p,q,-p,-q) + unknown_int()
```

In this template, `unknown_choice` is syntactically a Python function that must be specially interpreted by the tool. It takes k expressions as arguments and denoting that exactly one of these expressions must be used in the place of the `unknown_choice`. The goal of the synthesizer is to choose which of these expressions to use. Similarly, `unknown_int()` denotes a numeric constant to be selected by the synthesizer. In this example, the synthesizer chooses three expressions (one for each `unknown_choice`) and one integer constant. In the solution, the synthesizer sets the first choice to `q`, the second choice to `p`, the third one to the expression `-q` and the choice of the integer constant is `-1`.

Task

Your task is to implement a synthesizer that given a function f, and a template of an inverse function, constructs an inverse function f_{inv}. The input function f has n parameters that are integers in the range $[-1000, 1000]$ and outputs m integers. The generated inverse function must satisfy:

$$\forall \boldsymbol{x} \in [-1000, 1000]^n \qquad f_{inv}(f(\boldsymbol{x})) = \boldsymbol{x}$$

In this project, the synthesizer tool must be implemented using Python and the z3 SMT solver. The input and the synthesized functions are using only a subset of the Python programming language. In the used subset, all function parameters (of `f` and `f_inv`) are integers. Integer local variables can be used and assigned to (`=`). The language subset also includes integer constants, arithmetic operations (`+`, `-`, `*`, `/`) and `if` statements. In the conditions of the `if` statements, numeric expressions can be compared (`<`, `<=`, `>`, `>=`, `==`, `!=`), logical `and`, `or` and `not` can be used. Tuples can be used for assigning multiple variables simultaneously (e.g. `a,b = b,a` swaps the values of `a` and `b`). In order to be able to return multiple integers, a return statement returning a tuple of integers should be used (in case only one integer is returned, the return statement does not return a tuple).

All execution paths of the functions `f` and `f_inv` (even the ones that cannot be ever taken) must include a `return` statement. Return statements must return the same number of values and all values must be integers. In the provided code, we include an interpreter for the Python subset.

Template and prerequisites

You are given a template of a solution implemented in python named pysyn.py. The python tool must implement the three commands. One of the commands is already implemented. The other two are to be implemented and **will be graded**.

Commands

- `./pysyn.py eval <python_file> <data_file>`
  Given a python file that contains the function `f`, this command evaluates the function on the values provided in the given data file. Each row in the data file contains a space separated vector of integers $\boldsymbol{x}$. For each $\boldsymbol{x}$ in the input, this command outputs a space separated vector of integers $f(\boldsymbol{x})$.
  The tool must not output anything else on the standard output. This command is already implemented in the provided code. Use this interpreter as a reference for the supported features in the used Python subset.
- `/pysyn.py solve <python_file> <data_file>`
  The input python file contains a function `f`. Each row of the data file contains a space separated vector $\boldsymbol{y}$. The goal of this command is to find a vector $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = y$. Essentially, the task is to compute the values for an inverse function, without actually generating the code of the inverse function.
  The tool must not output anything else, but the vectors $\boldsymbol{x}$ on the standard output. One vector $\boldsymbol{x}$ must be printed per line for each input vector $\boldsymbol{y}$. If no vector $\boldsymbol{x}$ exists, such that $f(\boldsymbol{x}) = y$, the tool must output `Unsat` to the standard output. If there are multiple solutions, the tool must print only one of them.
- `/pysyn.py syn <python_file>`
  The input is a python file, which contains a function `f`, as well as a function template `f_inv`. The tool must output on the standard output python code

of a function `f_inv`, such that $\forall \boldsymbol{x} \in [-1000, 1000]^n$. $f_{inv}(f(\boldsymbol{x})) = \boldsymbol{x}$. If no such function exists, the tool must output `Unsat`. The tool should not output anything else on the standard output.
There are no requirements for the format of the produced function (e.g. extra brackets, etc) as soon as executing it with the interpreter of `eval` produces the desired result.

Requirements for the project submission

Each submission must include the program `./pysyn.py`, that does not depend on any libraries that are not present by default in Python 2.7 other than the `z3` library. We will install `z3` such that calling `import z3` from `./pysyn.py` will work. Your code must not create temporary files or read or write files other than the ones provided in the command line.

The submissions should also include 5 tests that you used to test your code. Each test contains a python file and an input data file for calling `solve` on the tool. The python file may also include a template for the `f_inv` function that the `syn` should be able use (some tests will be run only on `solve`, not on `syn` and it may not be possible to synthesize the inverse function). We have included an example test in the code.

Your project will be graded on tests provided by the TAs. The tests provided by one team can also be used to grade all teams. The team that has submitted valid tests that manage to trigger an error (subject to free interpretation by the TAs) in the programs of most other teams will receive 20% increase in the points from the project.

The grading schema may be non-linear. One failing test out of 100 does *not* mean you will get 99% of the points. Be sure to handle all corner cases. Be careful with variable assignments and `if` statements.

Projects should be submitted by email to `veselin.raychev@inf.ethz.ch` before May 17, 23:59:59 (the timezone in Zurich – CEST).

Installing Z3

You can obtain z3 from `https://github.com/Z3Prover/z3`. Take the latest version from March 2015 or later (it has MIT license, older ones do not have an open-source license).