Assignment 8

Exercise 1

You have heard about Rice's theorem several times. A computable partial function f is a partial function that can be implemented by some computer program κ, e.g., the factorial function $x \mapsto x!$. A property of computable partial functions is a predicate P over programs, such that P does not discirminate between programs implementing the same partial function, i.e., if κ_1 and κ_2 implement the same partial function, then $P(\kappa_1) \iff P(\kappa_2)$.

Theorem (Rice)**.** *A property P of computable partial functions (c.p.f.) is decidable iff it is* trivial*, i.e., either no c.p.f. has P or all c.p.f. have P.*

Why the theorem speaks about a property of functions and not about an arbitrary property of programs? Give an informal proof of Rice's theorem by reduction to the halting problem.

Hint: Show that any algorithm that decides a non-trivial property P can be converted to an algorithm that decides the halting problem, i.e., an algorithm that decides whether a given program halts for a given input.

Exercise 2

You are given the numeric programs:

1.
```
y = x
if x < 0:
    y = -x
assert y >= 0
```

2.
```
if x > y:
    z = x
else:
    z = y
assert x <= z and y <= z
```

Prove via symbolic execution that the assertions hold.

Exercise 3

You are given the numeric program:

```
x = 1
y = 1
while x < n:
    assert x == y
    x = x + 1
    y = y + y
```

Prove via symbolic execution that the assertion does not always hold.

Exercise 4

Suppose we execute the function `main` (see below) concolically with the two symbolic variables b_0 and e_0 for b and e and that we unroll loops at most twice. For the first concrete execution we assume that b and e are both 0.

```
int pow(int b, int e)
{
  int r = b;
  for (int i = 0; i < e; i++)
  {
    r = r * b;
  }
  return r;
}

void main(int b, int e)
{
  var r = pow(b, e);
  if (e % 2 == 0)
  {
    if (r < 0)
    {
      ERROR;
    }
  }
}
```

1. What is the path constraint that will be gathered during this first execution?

2. Negate the last conjunct in the path constraint and solve the resulting formula to generate a new input.

3. What is the path constraint that will be gathered when executing function `main` with the new input?

4. Repeat this process (1. run and record path constraint, 2. negate conjunct in path constraint and solve the constraint for a new input) until you find an execution that reaches the `ERROR` statement.

5. Compare your concrete inputs to the test cases that are generated by the concolic test-generation tool *Pex* when manually unrolling the loops in the original program. Go to `http://www.pexforfun.com/`, click on "New", and start from the following program:

```
using System;
using System.Diagnostics.Contracts;

sealed class __DoNotInstrumentAttribute : Attribute { }

// [__DoNotInstrument]
public static class Math
{
    public static int Pow(int b, int e) {
    int r = b;
    for (int i = 0; i < e; i++) {
        r = r * b;
    }
    return r;
    }
}

public class Program
{
    public static int Puzzle(int b, int e) {
    var r = Math.Pow(b, e);
    if (e % 2 == 0) {
        if (r < 0) {
        Contract.Assert(false);
        }
    }
    return r;
    }
}
```

6. Now, suppose that function `pow` was uninstrumented (e.g., because it was part of a native library). What is the path constraint that will be gathered during the first execution of function `main` (again with $b == 0$ and $e == 0$)?

7. Negate the last conjunct in the path constraint and solve the resulting formula to generate a new input.

8. What is the path constraint that will be gathered when executing function `main` with the new input?

9. Is it possible to reach the `ERROR` statement by repeating this process (1. run and record path constraint, 2. negate conjunct in path constraint and generate new input by solving the constraint)?

10. Compare your concrete inputs to the test cases that are generated by the concolic test-generation tool *Pex* when manually unrolling the loops in the original program. You can start from the same program as above and you should uncomment the line with the `[__DoNotInstrument]` attribute, which makes all methods in class `Math` uninstrumented.