Assignment 11 (solution)

1. For the flow-sensitive pointer analysis from the lecture, write down the abstract transformer for field updates.

   Solution: The notation $m[x \mapsto y]$ denotes the map m modified to yield y when applied to x. Then the weak-update transformer is:

   $$[\![p.f := x^l]\!](m_p, m_h) = (m_p, m'_h)$$
   $$\text{where } m'_h = m_h[a.f \mapsto m_h(a.f) \cup m_p(x) \mid a \in m_p(p)]$$

2. You are given the following program:

```
0:  c = newObject T;
1:  t = c;
2:  i = 0;
3:  while (i < count) {
4:    n = newObject T;
5:    c.f = n;
6:    c = n;
7:    i++;
8:  }
9:  c.f = t;
10: assert t != n;
```

   (a) Run the flow-sensitive pointer analysis from the lecture on it.
   Solution:

```
0:  c = newObject T;
    { t->{}, c->{A0}, n->{} }
1:  t = c;
    { t->{A0}, c->{A0}, n->{} }
2:  i = 0;
    { t->{A0}, c->{A0}, n->{} }
3:  while (i < count) {
      { t->{A0}, c->{A0, A4}, n->{A4}, A0.f->{A4}, A4.f->{A4} }
4:    n = newObject T;
      { t->{A0}, c->{A0, A4}, n->{A4}, A0.f->{A4}, A4.f->{A4} }
5:    c.f = n;
      { t->{A0}, c->{A0, A4}, n->{A4}, A0.f->{A4}, A4.f->{A4} }
6:    c = n;
      { t->{A0}, c->{A4}, n->{A4}, A0.f->{A4}, A4.f->{A4} }
7:    i++;
```

```
      { t->{A0}, c->{A4}, n->{A4}, A0.f->{A4}, A4.f->{A4} }
8:  }
    { t->{A0}, c->{A0,A4}, n->{A4}, A0.f->{A4}, A4.f->{A4} }
9:  c.f = t;
    { t->{A0}, c->{A0,A4}, n->{A4}, A0.f->{A0,A4}, A4.f->{A0,A4} }
10: assert t != n;
```

(b) Can you prove the assertion on line 10 using the results of the analysis?

Solution: No, since variables `t` and `n` could be both null.

3. Write a program for which the flow-sensitive pointer analysis from the lecture infers the following abstract state at the end of the program:

```
{ a->{A0}, b->{A0,A1}, A0.f->{A0}, A1.f->{A0} }
```

Solution:

```
0: a = newObject T;
   { a->{A0} }
1: b = newObject T;
   { a->{A0}, b->{A1} }
2: if (*) {
     { a->{A0}, b->{A1} }
3:   b = a;
     { a->{A0}, b->{A0} }
4: }
   { a->{A0}, b->{A0,A1} }
5: b.f = a;
   { a->{A0}, b->{A0,A1}, A0.f->{A0}, A1.f->{A0} }
```

4. Run both the flow-sensitive and the flow-insensitive pointer analysis on the following program:

```
0: a = newObject T;
1: b = a;
2: if (a == b) {
3:   b = newObject T;
4: } else {
5: }
```

Solution:

(a) Flow-sensitive pointer analysis:

```
0: a = newObject T;
   { a->{A0} }
1: b = a;
   { a->{A0}, b->{A0} }
2: if (a == b) {
     { a->{A0}, b->{A0} }
3:   b = newObject T;
```

```
     { a->{A0}, b->{A3} }
4: } else {
     { a->{A0}, b->{A0} }
5: }
   { a->{A0}, b->{A0,A3} }
```

(b) Flow-insensitive pointer analysis:

```
0: a = newObject T;
1: b = a;
2: if (a == b) {
3:   b = newObject T;
4: } else {
5: }
   { a->{A0}, b->{A0,A3} }
```

5. Suppose we execute the following function symbolically with the two symbolic variables b_0 and e_0 for `b` and `e`.

```
int pow(int b, int e)
{
  int r = b;
  for (int i = 0; i < e; i++)
  {
    r = r * b;
  }
  if (e % 2 == 0)
  {
    if (r < 0)
    {
      ERROR;
    }
  }
  return r;
}
```

(a) Enumerate all path constraints that reach the ERROR statement when unrolling the loop at most 2 times.

Solution:

```
PC_0 == !(0 < e_0) && (e_0 % 2 == 0) && (b_0 < 0)
PC_1 == (0 < e_0) && !(1 < e_0) && (e_0 % 2 == 0) && (b_0 * b_0 < 0)
PC_2 == (0 < e_0) && (1 < e_0) && !(2 < e_0)
        && (e_0 % 2 == 0) && (b_0 * b_0 * b_0 < 0)
```

(b) Use the concolic test-generation tool *Pex* to find inputs that satisfy these path constraints by manually unrolling the original program. Go to http://www.pexforfun.com/, click on "New", and start from the following program:

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

```
public class Program
{
  public static int Puzzle(int b, int e)
  {
    int r = b;
    for (int i = 0; i < e; i++)
    {
      r = r * b;
    }
    if (e % 2 == 0)
    {
      if (r < 0)
      {
        Contract.Assert(false);
      }
    }
    return r;
  }
}
```

Solution: Pex produces the following inputs for the path constraints:

- PC_0: `b_0 == int.MinValue, e_0 == 0`
- PC_1: no input
- PC_2: `b_0 == 1200235993, e_0 == 2`