Problem 1. Pointer Analysis

Perform pointer analysis on the following Java snippet, and answer the following questions.

```java
public class A {
    public A foo() {
        return new A(); // h1
    }
}

public class B extends A {
    public A foo() {
        return new C(); // h2
    }
}

public class C extends A {
    public A foo() {
        return new A(); // h3
    }
}

public class Main {
    public static void main(String[] args) {
        A a = new B(); // h4
        a = a.foo();
    }
}
```

1. What is the true set of allocations that `a` can refer to at any point during the program’s execution.
   
   `{h2, h4}`

2. What are the `pts` tuples inferred from this code in a context-insensitive, flow-insensitive pointer analysis? To get you started here’s one: `pts(a, h4).`
Problem 2. Binary Decision Diagrams

Draw the optimal (minimum number of nodes) BDD for the following expression:

\[ \text{exists}(x_0, (x_0 \text{ XNOR } x_2) \land (x_1 \text{ XNOR } x_3)) \]

Result should be a BDD containing only nodes of \( x_1 \) and \( x_3 \) (equal values leading to 1, non-equal values to 0)

1. What is the variable order you chose to create this BDD?
   There are multiple possible variable orderings, but one would be \( x_0, x_2, x_1, x_3 \)

2. What is the minimum number of nodes? 5 nodes (or 3 if you don’t count the 1 and 0 nodes - which are always there unless the BDD is a tautology/contradiction)

Problem 3. Path Sensitive Analysis With Satisfiability Modulo Theories

Solution:

```c
int func(int x1, int data[], int N) {
    int v, z;
    if (0 <= x1 && x1 < N) { // cond1
        if (x1 <= N/2) { // cond2
            x2 = 2 * x1;
        }
        x3 = phi(x1, x2);
        v = data[x3]; // line 7
        if (v >= 0 && v < N) { // cond3
            z1 = data[v]; // line 9
        } else {
            z2 = 0;
        }
        z3 = phi(z1, z2);
        if (z3 >= 0 && z3 < x) { // cond4
            mid = (z3 + x) / 2
        }
    }
    return v;  // return v
}
```
data[3] = data[mid];  // line 15
}
   ret1 = data[data[3]];  // line 17
} else {
   ret2 = data[0];  // line 19
}
return phi(ret1, ret2);

Note: you DID NOT need to submit the SSA code with your solution. This is just the explanation for the solution.

Solution:

(declare-fun x1 () (_ BitVec 32))
(declare-fun x2 () (_ BitVec 32))
(declare-fun x3 () (_ BitVec 32))
(declare-fun data1 () (Array (_ BitVec 32) (_ BitVec 32)))
(declare-fun data2 () (Array (_ BitVec 32) (_ BitVec 32)))
(declare-fun data3 () (Array (_ BitVec 32) (_ BitVec 32)))
(declare-fun N () (_ BitVec 32))
(declare-fun v () (_ BitVec 32))
(declare-fun mid () (_ BitVec 32))
(declare-fun z1 () (_ BitVec 32))
(declare-fun z2 () (_ BitVec 32))
(declare-fun z3 () (_ BitVec 32))
(declare-fun ret1 () (_ BitVec 32))
(declare-fun ret2 () (_ BitVec 32))
(declare-fun ret () (_ BitVec 32))

(declare-fun zero () (_ BitVec 32))
(declare-fun two () (_ BitVec 32))
(assert (= zero #x00000000))
(assert (= two #x00000002))

(declare-fun cond1 () Bool)
(declare-fun cond2 () Bool)
(declare-fun cond3 () Bool)
(declare-fun cond4 () Bool)

(assert (bvsge N zero))
(assert (= cond1 (and (bvsle zero x1) (bvsle x1 N))))
(assert (= cond2 (bvsle x1 (bvsdiv N two))))
(assert (= x2 (bvmul two x1))
(assert (= x3 (ite cond2 x2 x1))
(assert (= v (select data1 x3))
(assert (= cond3 (and (bvsge v zero) (bvslt v N))))

3
(assert (= z1 (select data1 v)))
(assert (= z2 zero))
(assert (= z3 (ite cond3 z1 z2)))
(assert (= cond4 (and (bvsge z3 zero) (bvslt z3 x3))))
(assert (= mid (bvsdiv (bvadd z3 x3) two)))
(assert (= data2 (store data1 z3 (select data1 mid))))
(assert (= data3 (ite cond4 data2 data1)))
(assert (= ret1 (select data3 (select data3 z3))))
(assert (= ret2 (select data3 zero)))
(assert (= ret (ite cond1 ret1 ret2)))

(define-fun out-of-bounds ((index (_ BitVec 32))) Bool
  (or (bvslt index zero) (bvsge index N)))

(push)
(assert (and cond1 (out-of-bounds x3))) ; line 7
(check-sat)
(get-model)
(pop)
(assert (not (and cond1 (out-of-bounds x3))))
(pop)
(assert (and cond1 cond3 (out-of-bounds v))) ; line 9
(check-sat)
(pop)
(assert (not (and cond1 cond3 (out-of-bounds v))))
(check-sat)
(pop)
(assert (and cond1 cond4 (out-of-bounds z3))) ; line 15 (data[z])
(check-sat)
(pop)
(assert (not (and cond1 cond4 (out-of-bounds z3))))
(check-sat)
(pop)
(assert (and cond1 cond4 (out-of-bounds mid))) ; line 15 (data[mid])
(get-model)
(pop)
(assert (not (and cond1 cond4 (out-of-bounds mid))))
(pop)
(assert (and cond1 (out-of-bounds z3))) ; line 17 (inner)
(check-sat)
(get-model)
(pop)
(assert (not (and cond1 (out-of-bounds z3)))))

(push)
(assume (and cond1 (out-of-bounds (select data3 z3)))) ; line 17 (outer)
(check-sat)
(get-model)
(pop)

(assert (not (and cond1 (out-of-bounds (select data3 z3))))))

(push)
(assume (and (not cond1) (out-of-bounds zero))) ; line 19
(check-sat)
(get-model)
(pop)

**Interpretation.**

- Line 7 can crash when \( x == N / 2 \)
- Line 9 cannot crash
- Line 15 read can crash due to an overflow of mid.
- Line 15 write cannot crash.
- Line 17 inner read can crash when the value at data[v] is out of bounds.
- Line 17 outer read can crash when the value at data[z] is out of bounds.
- Line 19 read can crash when \( N = 0 \).