CS243 Final
8:30AM – 11:30AM
March 17, 2014

The exam is open book/notes/laptop. We do not guarantee power or Internet access, however.

Answer all 9 questions on the exam paper itself. The total number of points is 180, i.e., one point per minute.

Write your name here: ____________________________________________

I acknowledge and accept the honor code.

(signed) ____________________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>180</td>
<td></td>
</tr>
</tbody>
</table>
Question 1: True/False (27 points) (1 point per correct T/F, 2 points per correct explanation)

Indicate whether each of the following statements is TRUE or FALSE. Write a short (1-2 sentence) explanation for each answer. Read each statement carefully! Some of them may be subtle or tricky.

a) If a forward dataflow solver is operating on an acyclic graph and visits the nodes in reverse-post-order, initializing the interior points to bottom will lead to the MFP (maximal fixed point) solution.

TRUE

Because the graph is acyclic, there are no back edges, so the initial value of the interior points will never be used.

If you wrote that the dataflow problem must be monotone with finite descending chains and selected FALSE, we gave you full credit.

b) If a dataflow problem is monotone (i.e. \( f(x \land y) \leq f(x) \land f(y) \)), then the maximal fixed point solution computed by a dataflow solver equals the meet-over-paths solution.

TRUE

The maximal fixed point solution equals the meet-over-paths solution if the problem is distributive (i.e. \( f(x \land y) = f(x) \land f(y) \)), not just monotone.

c) If it is possible to register allocate a piece of code using only two physical registers, that means that the interference graph must be bipartite.

TRUE

If the interference graph is bipartite, that means it can be split into two sets of nodes with no edges between nodes within the sets, only between nodes in different sets.

If you said that a disconnected graph is not bipartite and therefore selected FALSE, we gave you full credit.
d) If your compiler has an open-world assumption, it is not possible to discover the entire call graph before the program actually runs.

TRUE  FALSE

Code can be dynamically loaded or constructed at run time, so it is not possible to discover the entire call graph.

e) Performing partial method compilation can improve optimization opportunities on the common paths, as well as reduce compilation time.

TRUE  FALSE

Using partial method compilation can reduce compilation time by compiling less code. It can improve optimization opportunities on the common paths because e.g. branches from rare code into common code are not considered.

f) After running pointer analysis, if the vP relation has exactly one tuple vP('x', 'Object: main@12'), that means that variable 'x' can point to at most one object at execution time.

TRUE  FALSE

The allocation site 'Object: main@12' can represent multiple objects at run time, e.g. if the allocation site is in a loop or in a function that is called multiple times.
g) For a Boolean function with N inputs, you can always design a truth table such that there are at least $2^N$ nodes (including terminal nodes) in the smallest possible binary decision diagram.

**TRUE**    **FALSE**

The last two rows of a BDD can have at most two nodes each. A boolean function with 3 inputs can have at a BDD with at most 1+2+2+2=7 nodes, which is <8. In general the statement is false for all N>=3.

h) In a cloning-based context-sensitive analysis like the one covered in class, the number of clones you will need of a method is exactly the number of paths through the call graph to that method.

**TRUE**    **FALSE**

The cloning-based analysis clones methods after collapsing strongly-connected components. The number of paths through the call graph could be infinite due to recursive cycles.

i) In general, a trace-based garbage collector will do more work when there are more garbage objects to collect.

**TRUE**    **FALSE**

The run time of trace-based garbage collector is based on the number of reachable objects, not garbage objects.
Question 2: Partial Redundancy Elimination (18 points)

Below is a flow graph with entry B₁. There is one evaluation of expression $x+y$ and one place where that expression is killed (by assignment to $x$). There are no other statements in any of the blocks that involve $x$, $y$, or the expression $x+y$.

In this question, you will do the first three steps of the PRE algorithm discussed in class and in the text.

a) For which blocks $B$ is $x+y$ anticipated at the beginning? At the end? List those sets $IN[B]$ and $OUT[B]$ that contain $x+y$ when we compute anticipated expressions.

Answer:
$IN(B₁)$, $OUT(B₁)$, $IN(B₂)$, $OUT(B₃)$
b) For which blocks \( B \) is \( x+y \) “available” (in the sense used in the PRE algorithm) at the beginning? At the end? That is, this data-flow analysis is of the forwards/intersection type, with a transfer function that says \( \text{OUT}[B] \) contains those expressions that are not killed in \( B \) and are either in \( \text{IN}[B] \) or are anticipated at the beginning of \( B \). Give the sets \( \text{IN}[B] \) or \( \text{OUT}[B] \) that contain \( x+y \) for this data-flow analysis.

Answer:
\( \text{OUT}(B_1), \text{OUT}(B_2), \text{IN}(B_3) \). Note: a common error was to include \( \text{IN}(B_2) \). But \( B_3 \) is a predecessor of \( B_2 \), and surely \( x+y \) is killed in \( B_3 \).

c) For which blocks \( B \) is \( x+y \) in “earliest” of \( B \)?

Answer:
\( B_1 \) and \( B_2 \).
**Question 3:** Dataflow Frameworks (18 points)

It is often useful to have bounds on the range of values that a variable can have at a point in a program. We can analyze such ranges using the following dataflow framework.

$V$, the set of values for the framework, is the set of mappings $m$ from the variables of the program to the value UNK and pairs $[L,H]$, where $L \leq H$. Intuitively, if $m(x) = \text{UNK}$, then we know nothing about values variable $x$ may have. If $m(x) = [L,H]$, then we know that the value of variable $x$ at the point where the mapping applies is at least $L$ and at most $H$. It is possible for $L$ to be “minus infinity,” i.e., there is no known lower bound, and similarly, $H$ can be “plus infinity.”

$\land$, the meet operator, is defined as follows. If $m_1$ and $m_2$ are two mappings, then $m_1 \land m_2$ is that mapping $m$ such that for all variables $x$:

1. if $m_1(x) = [a,b]$, and $m_2(x) = [c,d]$, then $m(x) = [\min(a,c), \max(b,d)]$.
2. If one of $m_1(x)$ and $m_2(x)$ is UNK, then $m(x)$ equals the one that is not UNK.
3. If both $m_1(x)$ and $m_2(x)$ are UNK, then $m(x) = \text{UNK}$.

a) Under what conditions is $m_1 \leq m_2$? While this condition involves all variables, the condition for each variable is the same. Thus, you may write the condition in terms of a particular variable $x$. Note: you will not get full credit if you use “min” or “max” in your answer. There is a simpler way to describe the condition.

Answer:
For each variable $x$: either $m_2(x) = \text{UNK}$, or $a < c$ and $b \geq d$, where $m_1(x) = [a,c]$ and $m_2(x) = [b,d]$. Note: many people interpreted $[a,b]$ as a range. It is stated in the problem to be a pair. Thus, you cannot use operators like set containment on it without explaining what you are doing. In general there were many type errors in Problem three. People confused mappings with sets and tried to take set unions and other operations. People confused ranges with values of variables, tried to apply the transfer function to variables rather than mappings, and many other type mismatches.

**ERROR CODE:** C1 means that you did not consider the possibility that a value could be UNK rather than a pair (also used in part (e)).

b) What is the top element of the semilattice?

Answer:
The mapping $m$ such that $m(x) = \text{UNK}$ for all variables $x$. I can't believe how many people just answered "UNK." Again, that's a type error. UNK is a value, not a mapping.

**ERROR CODE:** C2 means you did not define a mapping, but only gave a value (also used in part (c)).
c) What is the bottom element of the semilattice?

Answer:
The mapping m such that m(x) = [-infty, +infty] for all x.

d) Describe an appropriate transfer function for a block that contains only the assignment a = 1.

Answer:
f(m)(a) = [1,1] and f(m)(x) = m(x) for all x other than a. A surprisingly common error involved imagining that the range of a prior to the assignment was still part of the range of a after assignment. The problem explains what we are doing with this framework, and explicitly says that m(a) represents the value of a "at a point." Thus, assignment is destructive, as always.

ERROR CODE: C3 means you did not explain what happens to variables other than a (also used in part (e)).

e) Describe an appropriate transfer function for a block that contains only the assignment a = b + c.

Answer:
f(m)(x) = m(x) for x other than a;
f(m)(a) = if (m(b) == UNK or m(c) = UNK) then UNK else [u+w, v+x], where m(b) = [u,v] and m(c) = [w,x];
Question 4: Software Pipelining (20 points)

Consider a machine with three resources and a loop with the following precedence constraints.

The resources used by each instruction are marked with an ‘X’ in the table below. Each instruction only uses resources in the cycle in which it is issued.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Resource 1</th>
<th>Resource 2</th>
<th>Resource 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>D</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>F</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
a. If we use only basic-block scheduling (no unrolling or pipelining), what is the best throughput achievable for this loop?

<table>
<thead>
<tr>
<th>Clock cycle</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>H</td>
<td></td>
<td>G</td>
</tr>
<tr>
<td>6</td>
<td>D</td>
<td></td>
<td>D</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b. What is the lower bound on the initiation interval as computed from the resource constraints? What is the lower bound on the initiation interval as computed from the precedence constraints?

Resource T = 4  
Precedence T = 1.5

c. Use the basic software pipelining algorithm described in class to schedule the loop. Schedule in the following order (A, C, E, F, H, B, D, G). For every operation, schedule it in the earliest (single iteration linear) legal cycle. Given a multi-instruction SCC, if you fail to schedule anything in the SCC, undo the entire SCC and start again with the first instruction of the SCC at a later cycle. Show both the schedule for a single iteration and the modulo reservation table.
d. Can you find a better schedule? If yes, show its schedule and modulo reservation table.

<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>E</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>G</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>D</th>
</tr>
</thead>
</table>
**Question 5**: Pointer Analysis (20 points)

Consider the following snippet of Java:

```java
public class Bazinga {
    public Bazinga a;
    public Bazinga b;

    public static void main(String[] args) {
        Bazinga x = new Bazinga(); // h1
        Bazinga y = new Bazinga(); // h2
        Bazinga z = new Bazinga(); // h3

        bar(x, y);
        bar(z, z);
        z.a = x;
    }

    public static void bar(Bazinga z1, Bazinga z2) {
        if (z1 == z2) {
            return;
        }
        z1.a = z2;
        baz(z1);
        baz(z2);
    }

    public static void baz(Bazinga z) {
        if (z.b != z)
            z.b = z;
        else
            baz(z.b);
    }
}
```

For reference, here are the Datalog rules for a **context-insensitive** analysis:

- `vP(v, h) :- vP_0(v, h).`
- `vP(v1, h) :- assign(v1, v2), vP(v2, h).`
- `hP(h1, f, h2) :- store(v1, f, v2), vP(v1, h1), vP(v2, h2).`
- `vP(v2, h2) :- load(v1, f, v2), vP(v1, h1), hP(h1, f, h2).`
a) What are the $hP$ tuples inferred from this code in a context-insensitive analysis? To get you started, we have given you one of the tuples.

\[
\begin{align*}
&hP(h1, a, h2) & &hP(h1, b, h1) \\
&hP(h1, a, h3) & &hP(h1, b, h2) \\
&hP(h3, a, h1) & &hP(h1, b, h3) \\
&hP(h3, a, h2) & &hP(h2, b, h1) \\
&hP(h3, a, h3) & &hP(h2, b, h2) \\
&hP(h1, b, h3) & &hP(h2, b, h3) \\
&hP(h3, b, h1) & &hP(h3, b, h2) \\
&hP(h3, b, h3) & &hP(h3, b, h3)
\end{align*}
\]

For reference, here are the Datalog rules for a context-sensitive analysis:

\[
\begin{align*}
&vP_C(\_v, h) :\neg \ vP(v, h). \\
&vP_C(c1, v1, h) :\neg \ \text{assign}_c(c1, v1, c2, v2), \ vP_C(c2, v2, h). \\
&hP(h1, f, h2) :\neg \ \text{store}(v1, f, v2), \ vP_C(c, v1, h1), \ vP_C(c, v2, h2). \\
&vP_C(c, v2, h2) :\neg \ \text{load}(v1, f, v2), \ vP_C(c, v1, h1), \ hP(h1, f, h2).
\end{align*}
\]

b) What are the $hP$ tuples inferred from this code in a context-sensitive analysis?

\[
\begin{align*}
&hP(h1, a, h2) \\
&hP(h3, a, h3) \\
&hP(h1, b, h1) \\
&hP(h2, b, h2) \\
&hP(h3, b, h3) \\
&hP(h3, a, h1)
\end{align*}
\]

c) How many copies of bar are made for a context-sensitive analysis? \underline{2}

d) How many copies of baz are made for a context-sensitive analysis? \underline{4}
Question 6: Program Analysis with Datalog (20 points)

In this question, you will write a Datalog analysis that builds on top of the pointer analysis we covered in class. Encryption keys are sensitive information and we want to make sure they are zeroed out after they are used. We want to find cases where an encryption key may not be correctly zeroed in the heap. In particular, an encryption key should never be stored in a String object, because in Java a String is immutable so it cannot be overwritten.

EDB relations you can use for this problem:

- cha(t:T, n:N, m:M) : calling method ‘n’ on object of type ‘t’ has target method ‘m’
- actual(s:S, i:I, v:V) : ‘v’ is the ‘i’th actual parameter at call site ‘s’
- invokes(s:S, m:M) : call site ‘s’ calls method ‘m’
- returns(m:M, v:V) : method ‘m’ stores its return value in variable ‘v’
- vPc(c:C, v:V, h:H) : variable ‘v’ may point to object ‘h’ under context ‘c’
- hType(h:H, t:T) : object ‘h’ has type ‘t’

We start by defining a relation fromString(h), which gives the heap objects that may be derived from a String object. More precisely, it contains all objects that may be returned from a method in the String class:

```
.fromString(h) :- cha(“String”, _, m), returns(m, v), vPc(_, v, h).
```

a) We want to calculate a relation sensitive(m), which contains the sensitive methods to which we do not want to pass an argument that was derived from a String. For the purposes of this problem, any method in the PBEKeySpec class is considered to be “sensitive”. Using one of the EDB relations above, write a Datalog rule to calculate the set of “sensitive” methods:

```
sensitive(m) :- cha(“PBEKeySpec”, __, m).
```

b) Next, we want to calculate a relation vulnerable(c,s), which says that under context ‘c’, call site ‘s’ can pass an object derived from a String object as a parameter to a sensitive method. Using relations from the EDB as well as the fromString and sensitive relations, write a Datalog rule to calculate the vulnerable relation:

```
vulnerable(c,s) :- invokes(s, m), sensitive(m), actual(s, __, v), vPc(c, v, h), fromString(h).
```
c) Now consider other Objects that may contain sensitive data, like char[]. We want to make sure that any char[] object that is passed to a sensitive method is also passed as the first argument to “Arrays.fill”. Use a Datalog rule to define a relation **sanitized(h)**, which are char[] objects that have been passed as the first argument to “Arrays.fill”. Then write a Datalog rule that finds char[] heap objects that are passed as an argument to a sensitive method, but are **not** passed as the first argument to “Arrays.fill”.

Hint: You can negate a relation by adding a ‘!’ character in front of it.

```
sanitized(h) :- hType(h, "char[]"), invokes(s,"Arrays.fill"), actual(s, 1, v), vPc(_,v,h).
unsanitized(h) :- hType(h, "char[]"), sensitive(m), invokes(s,m), actual(s,_,v), vPc(_,v,h), !sanitized(h).
```

d) The **unsanitized** relation may not be complete, meaning it may be missing some cases where you have a char[] object that is passed to a sensitive method without subsequently being passed to “Arrays.fill” to be cleared. Give two reasons why the **unsanitized** relation may not be complete. (One or two sentence descriptions will suffice.)

A few possible answers:

1. The analysis is flow-insensitive, so it does not say anything about ordering.
2. vPc() is a "may" point to relation, not "must" point to relation, so we may think an object is sanitized when it is not.
3. Heap objects in the analysis may represent multiple objects at run time, so a different object allocated at the same spot may have been sanitized than the one passed to the sensitive method.
Question 7: Data Dependence Analysis (20 points)

a) Consider the following loop:

```c
for (i=0; i<n; i++) {
    for (j=i; j<n; j++) {
        a[i][j] = a[i][j] + 1;
    }
}
```

Write the equations to figure out if the read and write overlap.

- \(0 \leq ir < n\)
- \(i \leq jr < n\)
- \(0 \leq iw < n\)
- \(i \leq jw < n\)
- \(ir = iw\)
- \(jr = jw\)

What are the direction vectors from the read to the write and from the write to the read?

(= =) from the read to the write
none from the write to the read

b) Given the following set of equations:

- \(2x + 3y + z \geq 5\)
- \(4x + 8y + 2z \geq 4\)
- \(x - y \leq 0\)

What equations are generated by the Fourier-Motzkin algorithm when doing the one step of eliminating \(x\) from the system?

The normalized equations are:

- \(4x \geq -6y - 2z + 10\)
- \(4x \geq -8y - 2z + 4\)
- \(4x \leq 4y\)

Combining 1 and 3 we get

- \(4y \geq -6y - 2z + 10\) or \(10y + 2z - 10 = 0\) or \(5y + z - 5 = 0\)

Combining 2 and 3 we get

- \(4y \geq -8y - 2z + 4\) or \(12y + 2z - 4 = 0\) or \(6y + z - 2 = 0\)
c) Traditional C does not allow multi-dimensional arrays of variable bounds. For dynamically sized arrays, programmers have to use single-dimension arrays and linearize the accesses. For example, rather than writing $a[i][j]$, they write $a[n*i+j]$.

1. Is such a reference affine and analyzable by the techniques discussed in class?

   no, $n*i$ is not linear

2. One possible technique is for the compiler to delinearize the array references for the purpose of data dependence analysis; e.g. treat $a[n*i+j]$ as if it was $a[i][j]$ and then treat each dimension as a separate equality constraint. Given the following loop:

   ```c
   for (i=0; i<n; i++) {
       for (j=0; j<n; j++) {
           a[n*i+j] = ...
       }
   }
   ```

   Is it legal to delinearize this loop?

   yes

   How about for the following:

   ```c
   for (i=0; i<m; i++) {
       for (j=0; j<m; j++) {
           a[n*i+j] = ...
       }
   }
   ```

   Is it legal to delinearize this loop?

   no because the compiler doesn't know that $m \leq n$
3. It turns out that the test for legality of delinearization can be mapped to solving an integer linear programming problem. Given this code:

```c
for (j=lb; j<ub; j++) {
    a[n*(i+c1) + c2*j] = ...
}
```

Write an equation or set of equations that must be true in order for delinearization to be legal.

- \( c_2 \cdot (u_b - l_b) \leq n \)

Full credit if you give the tighter restriction

- \( 0 \leq c_2 \cdot l_b \)
- \( c_2 \cdot u_b \leq n \)
**Question 8: Loop Optimization (20 points)**

a) Is it legal to interchange the following loop nest? Explain using direction vectors.

```plaintext
for i
  for j
    a[i][j] = a[i-1][j+1] * 2;
```

Direction vector = (>,<)

If we swap the direction vector becomes (<,>) Therefore not legal to interchange

b) Is it legal to vectorize the following loop? Explain using direction vectors.

```plaintext
for i
  a[i] = a[i+1] * 2;
```

Yes, it's legal to vectorize because there is no (positive) dependence from the write to the read, only from the read to the write (>)

c) Given the following loop nest. What is the minimum outer loop unrolling necessary to bring the number of loads plus stores down to less than or equal to the number of multiplies. You may interchange if you like (but tell us if you do).

```plaintext
for i
  for j
    a[i][j] = a[i][j] * b[j] + a[i][j-1] * b[j-1] + a[i][j+1] * b[j+1];
```

Interchange
Then unroll the outer j loop twice.

```plaintext
for j
  for i
    a[i][j] = a[i][j] * b[j] + a[i][j-1] * b[j-1] + a[i][j+1] * b[j+1];
    a[i][j+1] = a[i][j+1] * b[j+1] + a[i][j] * b[j] + a[i][j+2] * b[j+2];
```

We have two stores, four loads and six multiplications
**Question 9: Garbage Collection (17 points)**

a) Assume we are using an *incremental garbage collector*. The state of the heap is currently:

![Diagram](image)

That is, A.f points to B, B.f points to C, and C.f points to C. Assume A is part of the root set.

1. Assume there are no read, write, or transfer barriers, and the mutator can run interleaved with the garbage collector. Come up with a sequence of operations such that the garbage collector will incorrectly find that object C is garbage, i.e. C stays in the *unreached* state at the end of tracing. Steps 1-3 are already written down for you.

<table>
<thead>
<tr>
<th>Step</th>
<th>Garbage Collector</th>
<th>Mutator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>Move A from unscanned to scanned</td>
<td></td>
</tr>
<tr>
<td>Step 2</td>
<td>Scan A, Move B from unreached to unscanned</td>
<td></td>
</tr>
<tr>
<td>Step 3</td>
<td>Load B.f into pointer p</td>
<td></td>
</tr>
<tr>
<td>Step 4</td>
<td>Store p into A.f</td>
<td></td>
</tr>
<tr>
<td>Step 5</td>
<td>Store p into A.f</td>
<td></td>
</tr>
<tr>
<td>Step 6</td>
<td>Move B from unscanned to scanned</td>
<td>Overwrite B.f with null</td>
</tr>
<tr>
<td>Step 7</td>
<td>Scan B</td>
<td></td>
</tr>
</tbody>
</table>

2. Now assume we are using a write barrier to catch the locations that are written by the mutator. Under what step will the write barrier trigger?

   **Step 4 (Store p into A.f)**

At a minimum, which object(s) will need to be rescanned due to the write barrier triggering?

   **Object A**
b) Suppose we are using the train algorithm for incremental garbage collection. Let "car ij" refer to the jth car on the ith train. The remembered set for car 54 can include which of the following references? For each statement, circle the appropriate answer.

<table>
<thead>
<tr>
<th>Reference</th>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>A reference from an object in car 54 to an object in car 77</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>A reference from an object in car 52 to an object in car 54</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>A reference from an object in car 54 to an object in car 73</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>A reference from an object in car 58 to an object in car 54</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

c) Suppose we are using the train algorithm for incremental garbage collection. Let "car ij" refer to the jth car on the ith train. The object o in car 11 has at least one reference from objects in other cars. When we garbage collect car 11, object o could wind up in a number of different cars. Which cars could o be placed in if (a) we are not in "panic mode" (b) we ARE in "panic mode"? Circle the appropriate answer for each case.

<table>
<thead>
<tr>
<th>Car</th>
<th>Not in panic mode</th>
<th>In panic mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car 11</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
<tr>
<td>Car 12</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
<tr>
<td>Car 13</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
<tr>
<td>Car 14</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
<tr>
<td>Car 15</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
<tr>
<td>Car 21</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
<tr>
<td>Car 32</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
<tr>
<td>Car 42</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
<tr>
<td>Car 51</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
</tbody>
</table>