CS243 Midterm Examination
Winter 2014-2015

February 11, 2015
11:00 am - 12:15 pm

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

Duration: 75 minutes

Please do not post anything on Piazza till the solutions are put up on the class website.

Answer all 5 questions on the exam paper itself.

Write your name here: ____________________________________

I acknowledge and accept the Stanford honor code.

(signed) ____________________________________

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>
1. [12 points] True/False

Answer the following questions (true/false) with a brief explanation to support your choice. No points are given unless the explanation is correct.

a. When applied to a program whose max loop depth is $d$, a forward monotone data flow analysis that visits all nodes in reverse postorder is guaranteed to converge in at most $d+2$ iterations.

   False. Only if data propagates along acyclic paths.

b. A monotone forward data flow analysis must set $\text{OUT}[\text{Entry}]$ to the “top” of the semi-lattice, and a backward flow must set $\text{IN}[\text{Exit}]$ to the “bottom”.

   False. It can be any arbitrary lattice point based on the dataflow.

c. For a monotone forward flow framework, once the output of some basic block reaches bottom, the output of all its successors will also reach bottom.

   False. The transfer function can set it to anything.

d. Consider a program with $2n$ variables and a machine with only 2 registers, it is not possible to avoid spilling if the interference graph has over $n^2$ edges.

   True. Consider a fully connected bipartite graph with $n$ nodes in each side (every node is connected to all the nodes in the other side but no nodes in the same side), the graph has $n^2$ edges. Adding any additional edge would cause a spill. You can further prove that this bound is tight.
2. [10 points] Consider the following register interference graph:

![Register Interference Graph]

a. What is the minimum number of registers, \( R \), required to avoid spilling. Why?

4. We have a 4 clique B-E-D-F

b. Show a register allocation that uses \( R \) number of registers for this graph. You can label the nodes in the graph with the register number assigned.

- E - r1
- B - r2
- D - r3
- F - r4
- C - r1
- A - r2
- G - r2
3. [13 points] Consider the following control flow graph.

![Control Flow Graph]

a. Draw the immediate dominator tree for the flow graph above.

![Immediate Dominator Tree]

b. What are the natural loop(s) in the above flow graph?

<table>
<thead>
<tr>
<th>Back Edge</th>
<th>Natural Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-&gt;D</td>
<td>{D,E,G}</td>
</tr>
<tr>
<td>F-&gt;A</td>
<td>{A,B,C,D,E,F,G}</td>
</tr>
</tbody>
</table>
c. Is the flow graph reducible? Explain your answer.

No. Depending on how you construct your tree, the B->D or E->B edge becomes a retreating edge. The G->F edge however is a cross edge.
4. [15 points] Show the result of running partial redundancy elimination. What’s the final optimized flow graph? You don’t need to show the intermediate steps.
5. [25 points] Taint Analysis

Data from untrusted sources could cause security vulnerability in programs. For example, in C, a hacker can potentially take over the control of a program by passing well-crafted strings into a format function (e.g. printf()).

Consider a simplified language whose variables can only take string values and has the following kinds of statements:

- **ASSIGNMENT:** \( a = \text{<constant string>} \)
- **COPY:** \( a = b \)
- **CONCATENATION:** \( a = b+c \)
- **INPUT:** \( a = \text{read}() \)
- **PRINT:** \( \text{print}(a) \)
- **CONDITIONAL BRANCH:** \( \text{if}(a) \text{ goto } L \)

A variable is *tainted* if it is defined by an INPUT statement, or a CONCATENATION or COPY of other tainted variables. This problem has two parts:

a. Use data flow analysis to issue warnings whenever a PRINT statement may write out a tainted variable.

b. Indicate for each warning the INPUT statements that may be responsible for the warning.

For example, for the following code:

```
L1: a = \text{read}() \\
L2: c = \text{read}() \\
L3: \text{if}(a) \text{ goto } L5 \\
L4: a = c \\
L5: b = a + \text{"str"} \\
L6: \text{print}(c) \\
L7: \text{print}(b)
```

a. Warnings should be issued on both L6 and L7.

b. L6 warning: input statements that may be responsible: \{L2\};

   L7 warning: input statements that may be responsible: \{L1, L2\}

You can design one or more data flow analyses to answer the two parts of this problem. You will get partial credits by solving the first part of the problem. Below is a template that you need to fill out for each analysis.
<table>
<thead>
<tr>
<th>Property</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction of the analysis (forward/backward)</td>
<td>Backward</td>
</tr>
<tr>
<td>Meaning of the values in the semi-lattice</td>
<td>For each PRINT statement, keep a set of variables at a program point which have the following property, if the values of them at the point could be directly assigned from an INPUT statement, there should be a warning for that PRINT statement.</td>
</tr>
<tr>
<td>Semi-lattice diagram. (Label the top and the bottom elements)</td>
<td>For a single PRINT statement, the diagram is the same as live variable analysis (Union on set of variables). The semi-lattice diagram for a program is the product of the diagrams of all PRINT statements.</td>
</tr>
</tbody>
</table>
| Transfer function of a basic block | For simplicity, $f(m)$ is a transfer function for a single statement $s$. Let $m' = f(m)$. We define $f$ based on different statements. Note that $m' = m$ unless we explicitly specify what $m'$ is.  
1. Assignment $a = \text{<constant>}$. If $a \in m$, $m' = m \setminus \{a\}$.  
2. Copy $a = b$. If $a \in m$, $m' = m \setminus \{a\} \cup \{b\}$.  
3. Concatenation $a = b + c$. If $a \in m$, $m' = m \setminus \{a\} \cup \{b, c\}$.  
4. Input $a = \text{read()}$. If $a \in m$, $m' = m \setminus \{a\}$.  
5. Print $\text{print}(a)$. If this is the print statement we are calculating, $m' = m \cup \{a\}$. |
| Boundary condition. (assignment to OUT[entry]/IN[exit]) | $\text{IN[exit]} = \emptyset$ for each PRINT statement |
| Initialization for the iterative algorithm | $\text{IN[B]} = \emptyset$ for each PRINT statement |
| Is the framework monotone? (Yes/no: No explanation needed) | Yes |
| Is the framework distributive? (Yes/no: No explanation needed) | Yes |
| Will the algorithm converge? (Yes/no: No explanation needed) | Yes |

State explicitly how you generate the warnings and the causes of the warnings.
After running the data flow analysis, we go through the program one more time. For each INPUT, if the variable got assigned by the INPUT is in the variable set immediately after the INPUT corresponding to a PRINT, we put the line number of the INPUT in that PRINT set. We then issue warnings if there are any lines in the set after we go through the whole program.

NOTE: There’s another forward data flow analysis solution that can solve this problem, which is also acceptable.