This is an open-book, open-notes, open-laptop, closed-network, closed-classmate discussion exam.

You will have 1 hours and 30 minutes to complete the exam, and an additional 15 minutes to upload the exam for a total of 1 hour and 45 minutes. The examination has 4 problems. Please budget your time accordingly. Write your answers in the space provided on the exam. If you use additional scratch paper, please turn that in as well.

Your Name: ________________________ SUNet ID: ________________________

The following is a statement of the Stanford University Honor Code:

1. The Honor Code is an undertaking of the students, individually and collectively:
   
   (a) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
   
   (b) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

2. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

3. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

Signature: ________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>25</td>
<td>25</td>
<td>15</td>
<td>20</td>
<td>85</td>
</tr>
</tbody>
</table>

1
Problem 1. [25 Points]

Short answer questions:

1. (5 Points) True or False, briefly justify: If a node $n_1$ dominates another node $n_2$, then node $n_2$ is always visited before $n_1$ in a depth-first search.

   False. The dominance definition implies $n_1$ is visited before $n_2$.

   -1 if answered “true” but explanation correct.

2. (5 Points) Define $\gcd(a, b)$ to be the greatest common divisor of integers $a, b$, i.e., the largest positive integer that divides each of $a$ and $b$.

   Is $\gcd$ a valid meet operator over the set of positive integers? (Note: $\gcd(a, \gcd(b, c)) = \gcd(\gcd(a, b), c)$) If so, is there a top element, if so what is it? And is there a bottom element, if so what is it?

   True. Meet operator needs to be 1. idempotent 2. transitive 3. associative. NOTE: Proof not required!

   1, 2 are trivial. 3: $e = \gcd(a, \gcd(b, c)) \implies e \mid \gcd(b, c) \implies e \mid b \land e \mid c$, and $e \mid a$. This further implies $e \mid \gcd(a, b)$ thus $e \mid \gcd(\gcd(a, b), c)$.

   Conversely for $f = \gcd(\gcd(a, b), c)$ we can derive $f \mid \gcd(a, \gcd(b, c))$.

   Thus, $e = f$.

   There’s no top element. Bottom element is 1.

3. (5 Points) Suppose there’s a dataflow framework with set complement $\neg S$ as the transfer function, and set union $\cup$ as the meet operator. ($\neg S = U - S$, where $U$ is the universal set. You may assume the universal set is $\{1, 2, 3, 4, 5\}$)

   Is this a monotonic framework? Briefly explain.

   No. For meet $= \cup$, the $\leq$ relation is $\supseteq$, so $\{1, 2, 3\} \leq \{1, 2\}$, and $U - \{1, 2, 3\} = \{4, 5\}$, but $U - \{1, 2\} = \{3, 4, 5\}$, thus $f(\{1, 2, 3\}) \nleq f(\{1, 2\})$.
4. (5 Points) Write a possible reverse postorder traversal of this graph. Assume that A is the entry point and D is the exit.

List a postorder first: e.g., D, C, B, A. Then reverse: A, B, C, D.

Other possible postorders will be accepted.

In particular, C could also be the last element because of the self-loop.

5. (5 Points) Partial Redundancy Elimination: True/False (Justification not required)
Consider the following control flow graph (We have already inserted basic blocks along all edges that lead to nodes with multiple predecessors):

(a) (1 points) When we compute anticipated expressions, $a + b$ is in $Out[B2]$

Yes. Because the following block B4 uses it.
(b) (2 points) $a + b$ is in $\text{Earliest}[B2]$

Yes. Because $\text{Out}[B1]$ is not anticipating it.

(c) (2 points) $a + b$ is in $\text{Earliest}[B9]$

No. because $B9$ can go to $B13$, which doesn’t use $a + b$, so not anticipated.
Problem 2. Detecting divide-by-0 errors [25 points]

Your task is to design a dataflow analysis and issue warnings for potential divide-by-0 errors.

Your instruction set is as follows:

1. \( x = \text{input}(); \) // Integer input from user
2. \( x = y; \) // Assignment
3. \( x = y \times z; \) // Integer Multiplication
4. \( x = y / z; \) // Integer Division. Divide-by-0 error occurs when \( z \) is 0.
5. \( \text{assert } x; \) // Assert that \( x \) is not 0
6. \( \text{assert } !x; \) // Assert that \( x \) is 0
7. Branching instructions that do not mutate variables

You should be able to use the results of your dataflow analysis to issue warnings for the following two cases:

1. Issue a warning when you are guaranteed to divide-by-0
2. Issue a warning when you may divide-by-0

Please make the following assumptions:

- Each basic block holds exactly one of these instructions.
- All values are integers
- \( y/z \) will be non-zero unless \( y \) is 0
- During runtime if an assert fails, the entire program will exit. Note: This does not mean that an assert branches to the exit control flow node.

In the example: B5 is guaranteed to divide-by-0. B5 and B7 may divide-by-0 (A guaranteed to divide-by-0 implies a may divide-by-0, so B5 is also a may). B6 will never divide-by-0
(15 points) Fill out the dataflow analysis table below.

As with all dataflow problems, there are usually multiple solutions. This one can detect both the guarantee and may simultaneously.

<table>
<thead>
<tr>
<th>Direction of your analysis (forward/backward)</th>
<th>Forward.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice elements and meaning</td>
<td>A map between every variable and one of the following values: Undef (U), Zero (Z), Not Zero (NZ), Not-A-Constant (NAC). Same as the Constant Prop Lattice, but with only two 'constants' Z and NZ.</td>
</tr>
<tr>
<td>Meet operator or lattice diagram</td>
<td>Meet every variable individually. Undef &gt; Z, Undef &gt; NZ, Z &gt; NAC, NZ &gt; NAC</td>
</tr>
<tr>
<td>Is there a top element? If yes, what is it?</td>
<td>Every variable maps to Undef.</td>
</tr>
<tr>
<td>Is there a bottom element? If yes, what is it?</td>
<td>Every variable maps to NAC.</td>
</tr>
</tbody>
</table>
| Transfer function of a basic block             | x = input(): Out[b][x] = NAC  
x = y: Out[b][x] = In[b][y]  
x = y * z: out[b][x] = see table below  
x = y/z: out[b][x] = In[b][y]  
Assert x: out[b][x] = NZ  
Assert !x: out[b][x] = Z |
| Boundary condition initialization              | Bot |
| Interior points initialization                | Top |

<table>
<thead>
<tr>
<th>y \ x</th>
<th>U</th>
<th>Z</th>
<th>NZ</th>
<th>NAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>U</td>
<td>Z</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>NZ</td>
<td>U</td>
<td>Z</td>
<td>NZ</td>
<td>NAC</td>
</tr>
<tr>
<td>NAC</td>
<td>U</td>
<td>Z</td>
<td>NAC</td>
<td>NAC</td>
</tr>
</tbody>
</table>
(5 points) Describe how to use the results of your dataflow analysis to issue a warning for every instruction that is \textit{guaranteed} to divide-by-0:

For every divide instruction $x = y/z$, if $In[b][z]$ is $Z$, issue a warning.

(5 points) Describe how to use the results of your dataflow analysis (or create a new one) to issue a warning for every instruction that \textit{may} divide-by-0:

For every divide instruction $x = y/z$, if $In[b][z]$ is not NZ, issue a warning.
Alternate solution using two different dataflow analyses

(5 points) Describe how to use the results of your dataflow analysis to issue a warning for every instruction that is \textit{guaranteed} to divide-by-0:

| Direction of your analysis (forward/backward) | Forward. |
| Lattice elements and meaning | Set of variables. A variable being in the set represents the value must be 0 |
| Meet operator or lattice diagram | Intersection |
| Is there a top element? If yes, what is it? | Universal set |
| Is there a bottom element? If yes, what is it? | Empty set |
| Transfer function of a basic block | $x =$ input(): $Out[b] = In[b] - \{x\}$  
$x = y$: if $y$ in set add $x$ to set, if $y$ not in set remove $x$ from set.  
$x = y \times z$: if $y$ or $z$ in set, add $x$ to set. Else remove $x$ from set.  
$x = y/z$: Same as assignment (ignore $z$)  
\textbf{Assert} $x$: $out[b] = In[b] - \{x\}$  
\textbf{Assert} $!x$: $out[b][x] = In[b] \cup \{x\}$ |
| Boundary condition initialization | Empty set |
| Interior points initialization | Universal set |

For every divide instruction $x = y/z$, if $z \in In[b]$ issue a warning
(5 points) Describe how to use the results of your dataflow analysis (or create a new one) to issue a warning for every instruction that may divide-by-0:

<table>
<thead>
<tr>
<th>Direction of your analysis (forward/backward)</th>
<th>Forward.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice elements and meaning</td>
<td>Set of variables. A variable being in the set represents the value must not be 0</td>
</tr>
<tr>
<td>Meet operator or lattice diagram</td>
<td>Intersection</td>
</tr>
<tr>
<td>Is there a top element? If yes, what is it?</td>
<td>Universal set</td>
</tr>
<tr>
<td>Is there a bottom element? If yes, what is it?</td>
<td>Empty set</td>
</tr>
</tbody>
</table>
| Transfer function of a basic block            | **x = input()**: $Out[b] = In[b] - \{x\}$  
**x = y**: if y in set add x to set, if y not in set remove x from set.  
**x = y * z**: if y or z not in set remove x from set. Else add x to set.  
**x = y/z**: Same as assignment (ignore z)  
**Assert x**: $out[b] = In[b] \cup \{x\}$  
**Assert !x**: $out[b][x] = In[b] - \{x\}$ |
| Boundary condition initialization             | Empty set |
| Interior points initialization                | Universal set |

For every divide instruction $x = y/z$, if $z \notin In[b]$ issue a warning
**Problem 3.** Register Allocation [15 points]

Consider the following control flow graph:

![Control Flow Graph]

You can safely assume that the variables A, B, C, D are not used or defined in any "...

(10 pts) Execute the register allocation algorithm based on coloring, as shown in class. Show the interference graph, and each step of the algorithm.

The interference graph is complete minus the (A, C) edge. A does not interfere with C because the beginning of their live ranges do not interfere. Many possible solutions for the register allocation coloring. For example, using the above sequence you could get: \( D = 1, C = 2, B = 3, A = 2 \)

(5 pts) Can you allocate with 2 registers? If not, show a safe code transformation (ie. such that the overall behavior is not changed) so that it can be allocated with 2 registers without spilling.

You cannot allocate this CFG with 2 registers. Swapping the two instructions in B3 removes the (B, D) edge in the interference graph and allows for register allocation using 2 registers.
**Problem 4.** Binary Decision Diagrams [20 points]

Consider the Call Graph below. Recall that the nodes represent functions and a directed edge from A to B represents that function A could call function B.

1. (10 pts) Your first task is to represent the call graph as a Reduced Ordered Binary Decision Diagram (ROBDD) for a *context insensitive* analysis. Encode the states \((A, B, C, D)\) with the values \((00, 01, 10, 11)\) respectively. Use the BDD variables \((X_0, X_1, X_2, X_3)\) to represent that \((X_0, X_1)\) calls \((X_2, X_3)\). For example “A calls B” is represented by \(X_0, X_1, X_2, X_3 = 0, 0, 0, 1\). Draw the final ROBDD below. Assume the variables are in the fixed order: \((X_0, X_1, X_2, X_3)\).
2. (5 pts) Your second task is to construct another BDD computing “The formula representing calls to the C or D function”. Write out the formula using any combination of the **Apply**, **Restrict**, **Exists** operators on your previously computed BDD. Briefly justify. You do **not** need to evaluate or draw the resulting BDD, although it might help in verifying your answer. *Hint:* Use your knowledge regarding how (A,B,C,D) are encoded in part 1.

Calls to C or D requires constraining the x2 and x3 variables to only be valid for C or D.
Will accept anything equivalent to Restrict(BDD, X_2 = 1)
For example:
Apply(OR, Restrict(BDD, X_2 = 1, X_3 = 0), Restrict(BDD, X_2 = 1, X_3 = 1))

3. (5 pts) The previous parts were assuming a context-insensitive analysis on the given call graph. Draw the expanded call graph for a **context-sensitive** analysis. What is the minimum number of BDD variables required to encode the context numbering for this expanded call graph?

Since there are 8 states in the expanded call graph, this requires at least 3 bits to encode each state. Thus requiring 4 variables to encode the call graph (3 for caller 3 for callee)