This is an open-book, open-notes, open-laptop, closed-network exam. Please do not post anything on Piazza until the solutions are put up on the class website.

You have 3 hours to work on this exam. The examination has 7 problems worth 165 points. 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>#5</th>
<th>#6</th>
<th>#7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>30</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>165</td>
</tr>
</tbody>
</table>
Problem 1. True or false. Specify clearly whether the following statements are true or false. You must justify your answers in 1 to 5 lines [30 Points].

1. Global scheduling, as discussed in class, can potentially slow a program down.

   True. Global instruction scheduling can add more instructions to a path that is anticipated to be executed less frequently. If this prediction is incorrect and the path is taken more often, it could lead to worse performance.

2. Data-flow in Datalog

   (a) We can formulate the reaching definitions problem using Datalog.

   True. We can have three tuples, \( \text{def}(l, x) \), meaning variable \( x \) is defined at line \( l \); \( \text{reach}(l, x, l') \), meaning the definition of \( x \) at \( l \) reaches line \( l' \); and \( \text{suc}(l, l') \), meaning \( l' \) is an immediate successor of \( l \) in the control flow graph. Our goal is to find all the \( \text{reach} \) tuples with the following two rules:

   \[
   \text{reach}(10, x, 11) \leftarrow \text{def}(10, x), \text{suc}(10, 11) \\
   \text{reach}(10, x, 12) \leftarrow \text{reach}(10, x, 11), \text{suc}(11, 12), \neg \text{def}(11, x)
   \]

   (b) If the answer is true above, then if we use a BDD implementation of Datalog to find reaching definitions, it will run faster than the data flow algorithm described in class that iterates the graph in reverse postorder. False. BDD is slower because each BDD operation will only move the data flow information one basic block at a time. Whereas reverse postorder will finish in just a few iterations.

3. For each of the metric below, is it true or false that generational garbage collection improves over the Baker’s algorithm:

   (a) Space reclaimed False. Generational GC will clean-up younger objects more frequently, but it will eventually reclaim all dead objects.

   (b) Overall execution time True. As the root set already includes all pointers from earlier generations, and we expect there not to be too many references into the generation being collected (as young objects die quickly), less time will be spent finding reachable objects, reducing the overall execution time.

   (c) Space usage False. There is more space overhead to organize objects into generations. Also, older objects that die are cleaned up less frequently, so until a full scan is performed they can be occupying space.

   (d) Pause time True. By only collecting when a partition fills up, it will run more frequently and thus take less time per collection.

   (e) Data locality True. Objects are partitioned based on age, and as young objects typically die younger this leads to larger free regions and less fragmentation.

4. Different depth-first traversals of a reducible flow graph may yield different retreating edges.
False. In a reducible flow graph, retreating edges must be back edges. Therefore, if \( y \to x \) is a retreating edge in some depth-first traversal, \( x \) must dominate \( y \) in the control flow graph, which means in any depth-first traversal, \( x \) must be an ancestor of \( y \).

5. For the expression \((\neg x_1 \ OR \ x_2) \ AND \ (x_1 \ XOR \ x_2)\), it is possible to create a BDD with just 4 nodes (including the 0 and 1 nodes).

True, it is possible \((x \ is \ x_1, \ y \ is \ x_2)\):

\[
\begin{array}{c}
\text{x} \\
\downarrow \\
\text{y} \\
\downarrow \\
\text{1} \\
\end{array}
\]
Problem 2. Pointer Analysis [20 points]

Perform a context-insensitive, flow-insensitive pointer analysis on the code below. List the hpts tuples that are produced by the analysis.

```java
public class A {
    public A f;
    public void foo(A p, A q) {
        p.f = q;
    }
    public void bar() {
        this.f = new A(); // h1
    }
}

public class Main {
    public static void main(String[] args) {
        A a = new A(); // h2
        A b = new A(); // h3
        b.bar();
        a.foo(a, b);
        b = b.f;
    }
}
```

hpts(h1, f, h1)
hpts(h2, f, h1)
hpts(h2, f, h3)
hpts(h3, f, h1)
Problem 3. Partial Redundancy Elimination [20 points]
Solution:
Problem 4. Software Pipelining [20 points]

Consider the following data dependency graph. Next to each instruction are indicated the resources used by that instruction.

1. What is the minimum initiation interval from the precedence constraints?
   
   3 from top loop, 2 from the bottom loop, so 3 overall.

2. What is the minimum initiation interval from resource usage?
   
   3 for each resource type, so 3 overall.

![Data Dependency Graph](image-url)
3. Find the optimal software pipelined schedule for this loop. Show both the modulo reservation table and the generated code (with nops if necessary) for a single iteration of the loop.

The minimum initiation interval is 3. Modulo reservation table:

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>E</td>
<td>B</td>
</tr>
</tbody>
</table>

Single iteration: A, nop, B, nop, nop, nop, C, nop, D, nop, E

4. Can this schedule be generated using the heuristic algorithm described in class? Why or why not?

Yes, as it will first attempt to place C in the middle left spot, but it will not be able to successfully schedule the rest of the SCC, so it will then attempt to place C in the top left spot which will lead to a valid schedule.
Problem 5. Move Semantics [25 points]

Some programming languages offer special “move” instructions to support transferring of resource ownership between objects or pointers. An example is the unique_ptr (unique pointer) in C++. We say that a unique_ptr is valid if it points to a resource, and invalid otherwise. At any given time, there is one and only one (valid) unique_ptr that refers to a resource. All unique_ptr pointers are invalid at the beginning of the program. A make instruction creates and returns a resource that can be assigned to a pointer. A move instruction transfer the ownership from the source to the destination pointer; the source unique_ptr can no longer be used, as the move renders it invalid. When the pointer is overwritten or goes out of scope, if it holds a valid resource, the resource is automatically destroyed by the run-time system.

There are two core instructions that the programmer writes:

- \( x = \text{make} (\text{Resource}()) \);
  creates a resource object and gives ownership to the unique_ptr \( x \).

- \( y = \text{move} (x) \);
  moves the underlying resource out of the unique_ptr \( x \) and into the unique_ptr \( y \).

The compiler is responsible for inserting instructions to destroy resources. There are two versions of such instructions:

- \( \text{destroy} \_\text{if\_valid} (x) \);
  destroys the underlying resource if \( x \) is valid.

- \( \text{destroy} (x) \);
  destroys the underlying resource unconditionally, expecting \( x \) to be valid.

Automatic destruction is implemented with a straightforward compiler pass that inserts a destroy\_if\_valid\( (x) \) instruction:

1. right before any \( x = \text{make} (\text{Resource}()) \) instruction
2. right before any \( x = \text{move} (y) \) instruction, where \( x \neq y \)
3. right before variable \( x \) goes out of scope.

You have two tasks:

1. Every \( x = \text{move} (y) \) instruction requires \( y \) to be valid. Warn if it is possible that \( y \) may be invalid.

2. Optimize the destruction of the resources by removing unnecessary destroy\_if\_valid\( (x) \) instructions and by replacing destroy\_if\_valid\( (x) \) instructions with the faster version destroy\( (x) \), whenever possible.
For example:

destroy_if_valid(a); // REMOVE, as a is not valid
da = make(Resource());
destroy_if_valid(b); // REMOVE, as b is not valid
db = make(Resource());
destroy_if_valid(b); // CHANGE to destroy(b), as b is valid
b = move(a);
destroy_if_valid(c); // REMOVE, as c is not valid
c = move(a); // WARN HERE, as a is invalid

Describe your data-flow analysis by specifying the following:

- Direction of your data-flow analysis (forward/backward)
- Elements in the semi-lattice and their meaning
- Meet operator or lattice diagram
- Top and bottom elements if they exist
- Transfer function
- Boundary condition initialization
- Interior points initialization

Make sure you specify how you warn on uses of invalid objects and perform the destroy_if_valid(x) replacements using your analysis results.

Write your data-flow analysis answer here (may use the next page as well):
Continue your data-flow analysis answer here (if needed):

**Direction:** Forward

**Elements:** For each pointer, one of the following: {UNDEF, VALID, INVALID, UNKNOWN}

**Meet operator:**

(identity/commutative combinations can be inferred)

UNDEF + VALID = VALID
UNDEF + INVALID = INVALID
UNDEF + UNKNOWN = UNKNOWN
VALID + INVALID = UNKNOWN
VALID + UNKNOWN = UNKNOWN
INVALID + UNKNOWN = UNKNOWN

**Top:** All pointers set to UNDEF.

**Bottom:** All pointers set to UNKNOWN.

**Transfer function:**

\[ f[b] = \begin{cases} \{x = \text{VALID}\} & \text{if } b \text{ is } x = \text{make}(\text{Resource}) \\ \{x = \text{INVALID}, y = \text{In}[b][x]\} & \text{if } b \text{ is } y = \text{move}(x) \ (x \neq y) \\ \text{In}[b] & \text{if } b \text{ is other} \end{cases} \]

**Boundary:** Out[Entry] = All pointers set to INVALID.

**Interior point initialization:** \( \top \)

For each instruction \( b = y = \text{move}(x) \), if In[b][x] = INVALID or UNKNOWN then warn on this usage. For each instruction \( b = \text{destroy\_if\_valid}(x) \), if In[b][x] = INVALID then remove, if In[b][x] = VALID then change to destroy(x), else (i.e. if In[b][x] = UNDEF or UNKNOWN) then leave as-is.
Problem 6. Parallelization [25 points]

Consider the following program.

\[
\begin{align*}
\text{for } (\text{int } i = 1; \ i < N; \ i++) \{ \\
\quad & A[i, i] = B[i, i]; \ \\
\quad & \text{for } (\text{int } j = i + 1; \ j < N; \ j++) \{ \\
\quad & \quad B[i, j] = (B[i-1, j] + B[i, 0]) \times A[i, j-1]; \\
\quad & \}\}
\end{align*}
\]

1. Draw the iteration space for each statement in the program. Use arrows to mark data-dependencies between iterations.

![Iteration Space Diagram](image)

2. Is there any communication-free parallelism that can be obtained using affine transformations? If yes, what is the affine partitioning for statement (1), and for statement (2)?

3. Assuming \( p \) is the processor ID, what is the range of \( p \)? What is the SPMD code for each processor? You do not need to optimize the code generated; simply wrap statements with the right conditional statements.

Solution:

Unoptimized:

\[
\begin{align*}
\text{for } (\text{int } p = 1; \ p < N; \ p++){ \text{ //Parallelizable} \\
\quad & \text{for } (\text{int } i = 1; \ i < N; \ i++)\{ \\
\quad & \quad \text{if } (i == p) \\
\quad & \quad \quad A[i, i] = B[i, i]; \\
\quad & \quad \text{for } (\text{int } j = i + 1; \ j < N; \ j++)\{ \\
\quad & \quad & \text{if } (p == j - 1)\{ \\
\quad & \quad & \quad B[i, j] = (B[i - 1, j] + B[i, 0]) \times A[i, j - 1]; \\
\quad & \quad & \}\}
\quad & \}\}
\end{align*}
\]
Optimized version:
The loop bound for statement (1)
1 <= p < N; p == i
The loop bound for statement (2)
1 <= i < N
j == p+1
i+1 <= p+1 < N
i <= p < N-1

Optimized code:
for (int p = 1; p < N; p++) {
    for (int i = 1; i < N; i++) {
        if (i==p) A[i,i] = B[i,i];
        if (i <= p) & (p < N-1)
            B[i,p+1] = (B[i-1,p+1] + B[i,0])* A[i,p];
    }
}

Further optimization:
for (int p = 1; p < N; p++) {
    for (int i = 1; i < min(p,N-1); i++) {
        B[i,p+1] = (B[i-1,p+1] + B[i,0])* A[i,p];
    }
    if (p < N-1)
        B[i,p+1] = (B[i-1,p+1] + B[i,0])* A[i,p];
}
Problem 7. Path sensitive analysis with SMT [25 points]

Consider the following function.

```c
1 double foo( double a, double b ) {
2    double c, d;
3    if ( b > 0 ){
4        c = 2 * a - 3 * b;
5        if ( c > b ){
6            d = c;
7        } else {
8            d = 0;
9        }
10        return c - d;
11    }
12    return 0;
13 }
```

1. Rewrite the program in SSA form.

Solution:

```c
phi0 = b0 > 0;
c0 = 2 * a0 - 3 * b0;
phi1 = c0 > b0;
d0 = c0;
d1 = 0;
d2 = phi1 ? d0 : d1;
r0 = c0 - d2;
r1 = 0;
r2 = phi0 ? r0 : r1;
```
2. Consider the following property $P$: the return value in line 10 must be less than the first input to the function (double $a$). Write an SMT formula to check whether this function has this property. You do not need to solve the formula or check whether the function actually has this property.

For simplicity, use Real to model double. First introduce the variables using declare-const (e.g., (declare-const $a_0$ Real)). Then write all the assertions for checking the property (e.g., (assert (= $\phi_0$ (> $b_0$ 0))))

You will not be deducted points for minor syntax errors.

Solution:

(declare-const $a_0$ Real)
(declare-const $b_0$ Real)
(declare-const $c_0$ Real)
(declare-const $d_0$ Real)
(declare-const $d_1$ Real)
(declare-const $d_2$ Real)
(declare-const $r_0$ Real)
(declare-const $\phi_0$ Bool)
(declare-const $\phi_1$ Bool)

(assert (= $\phi_0$ (> $b_0$ 0)))
(assert (= $c_0$ (- (* 2 $a_0$) (* 3 $b_0$))))
(assert (= $\phi_1$ (> $c_0$ $b_0$)))
(assert (= $d_0$ $c_0$))
(assert (= $d_1$ 0))
(assert (= $d_2$ (ite $\phi_1$ $d_0$ $d_1$)))
(assert (= $r_0$ (- $c_0$ $d_2$)))
(assert $\phi_0$)
(assert (>= $r_0$ $a_0$))

3. Suppose CVC4 returns $R$ on this formula, where $R \in \{\text{SAT, UNSAT}\}$. Describe how you can use $R$ to determine whether the function has property $P$?

The function has property $P$ iff $R$ is UNSAT.