CS243 Final Examination

Winter 2019

March 22, 2019

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 160 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:

a. The Honor Code is an undertaking of the students, individually and collectively:

   (i) 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;

   (ii) 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.

b. 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.

c. 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: ______________________

Problem	#1	#2	#3	#4	#5	#6	#7	**Total**
Score								
Max	20	20	20	20	25	25	30	160

Problem 1. True or False? Briefly justify your answer in 1 to 5 lines [20 Points]

a. When moving an instruction in a downward code motion, if the source block dominates the destination block, but the destination block does not post-dominate the source block, an extra operation may be executed.

b. In partial garbage collection, none of the garbage in the stable set is collected, but all the garbage in the target set is collected.

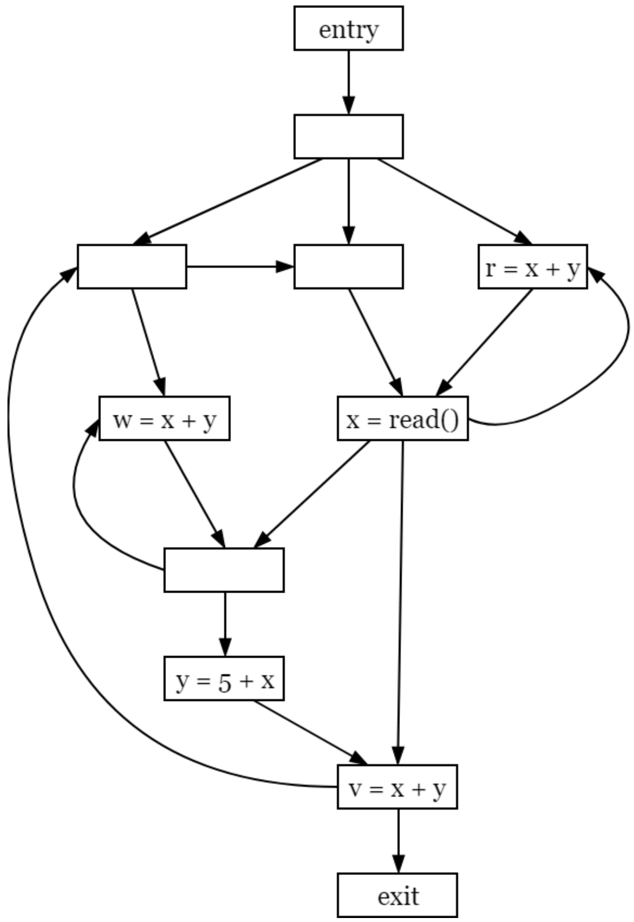
c. All primitive operations (Apply, Restrict, Exists) on Binary Decision Diagrams can be performed in time polynomial on the size (number of nodes) of the BDD.

d. There is no disadvantage to allocating registers before global instruction scheduling.

e. For the expression (x_1 `OR` x_2) `AND` ($\neg x_1$ `OR` $\neg x_2$), it is not possible to create a BDD with just 4 nodes (including the 0 and 1 nodes).

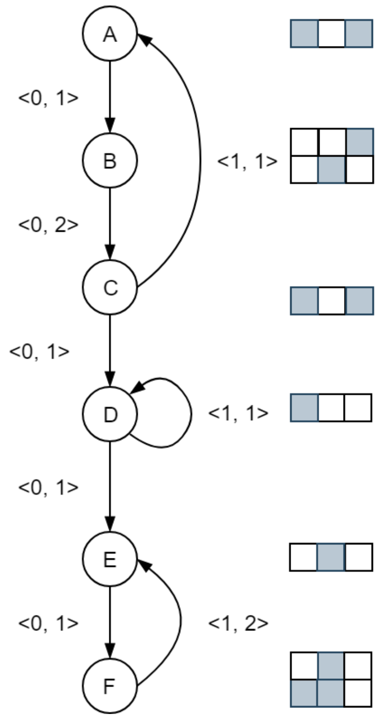
Problem 2. Partial Redundancy Elimination [20 Points]

Show the result of applying partial redundancy elimination to the following program. You do not need to show the intermediate steps.



Problem 3. Software Pipelining [20 Points]

In the following data dependence graph of a loop, each node is labeled with its resource reservation table, and each dependence edge is labeled with a tuple: the iteration difference and the delay.



a. What is the lower bound imposed on the initiation interval by resource constraints?

b. What is the lower bound imposed on the initiation interval by precedence constraints?

c. Find the optimal software pipelined schedule for this loop. What is the minimum initiation interval? Show the modulo reservation table and the code for a single iteration of the pipelined loop. (Show `nops` if necessary).

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

Problem 4. Pointer Analysis [20 Points]

Perform a context-insensitive, flow-insensitive pointer analysis on the code below. The call graph should be constructed on-the-fly as the algorithm discovers the points-to relationships. List the `hP` tuples that are produced by the analysis.

```
public class A {
    public A x;
    public void foo() {
        this.x = new A(); // h1
    }
}

public class B extends A {
    public void foo() {
        this.x = new B(); // h2
    }
}

public class Main {
    public static void main(String [] args) {
        A a = new B(); // h3
        a.x = new A(); // h4
        a.x.foo();
        a.foo();
    }
}
```

Problem 5. Smart Dead Code Elimination [25 Points]

The data flow analysis algorithms discussed in class ignore the value of conditions in the control flow graph. This problem explores how we can improve dead code elimination by analyzing the value of conditions.

Consider a programming language with the following operations: (`x`, `y`, `z` below are variables.)

- An assignment of the form "`x = true;`" or "`x = false;`".
- A boolean operation of the form "`x = not y;`" or "`x = y && z;`," or "`x = y || z;`" with obvious semantics.
- An assertion of the form "`assert(x);`" or "`assert(not x);`" asserts that the value of variable `x` is true or false, respectively. The program terminates if the assertion fails; thus the assertion can be assumed to be true if the execution proceeds to the next statement.
- A condition of the form "`if (x)`". Each condition has two control edges: one is labeled `true`, taken when `x` is true, and the other labeled `false`, taken when `x` is false.
- An unconditional "`branch`" statement which has a single control flow edge leading to the successor basic block.

Your task is to design a compiler that eliminates dead code, including the removal of unnecessary conditional statements, by statically determining if certain edges can never be taken. Assume that all variables are live at the end of the program, and every statement is in its own basic block.

For this problem, you may change the control flow graph, add instructions, and apply multiple passes of data flow analysis. If you are using a known data flow analysis, you need to mention only its name; if you are defining a new data flow analysis, answer the questions below. Do NOT use SMT in your solution.

a. What is the direction of the analysis?

b. What is the domain and boundary condition?

c. Describe the meet operator.

d. Describe the transfer function.

e. Is your data flow framework monotone?

f. Is your data flow framework distributive?

(This page is intentionally left blank).

Problem 6. Affine Transforms/Parallelization [25 Points]
Consider the following code:

```
for (i=1; i<n; i++) {
    A[i,i] = B[i];
    for (k=i+1; k<n; k++) {
        A[i,k] = A[i-1,k] / A[i,i];
    }
}
```

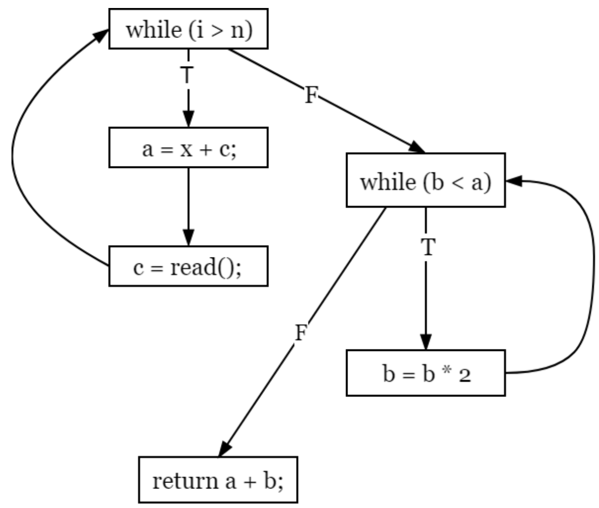
a. Is there any communication-free parallelism that can be obtained using affine transforms? If yes, show the result. You need not optimize the code generated; simply wrap all the code in a conditional statement.

b. Find the outermost fully permutable loop nest in this program. You need not optimize the code generated; simply wrap all the code in a conditional statement.

(This page is intentionally left blank).

Problem 7. While Loops [30 Points]
Your task is to create 2 tools that warn if a `while` loop may not terminate. You may assume that the `while`⟨`cond`⟩ loop has only one exit, which is taken when the ⟨`cond`⟩ evaluates to `false`.

a. Your first tool, T1, will warn about a possible infinite loop if none of the variables used in ⟨`cond`⟩ is updated in the loop. Consider the following example; each edge is annotated by the value of the condition evaluated in the source block.



For this example, a warning should be issued only for `while(i > n)`, as neither `i` nor `n` is updated within the loop.

You should generate a warning for *each* `while`⟨`cond`⟩ that may not terminate according to the above definition. **You may use any concepts, other than SMT, learned in this course here. You will have a chance to use SMT in parts b,c,d of this question.** If you are using a known analysis, you must specify clearly which analysis it is and how you use it in this problem. If you are defining a dataflow algorithm, answer the following questions:

(i) What is the direction of the analysis?

(ii) What is the domain and boundary condition?

(iii) Describe the meet operator and transfer function.

(iv) Is your data flow framework monotone?

(v) Is your data flow framework distributive?

(This page is intentionally left blank).

b. Your task is to build a different tool T2 using SMT. Consider this following program:

```
while(i < N) {
   i--;
   prev = data[i];
   i++;
   diff[i] = data[i] - prev;
}
```

We see that the value of `i` is updated within the loop, but its value remains unchanged at the end of each iteration. Unroll the loop twice, and write the SMT formula that checks if the condition tested at the end of two iterations may differ from the condition evaluated upon entry of the loop.

c. Now consider applying the same approach to an *arbitrary* loop, what can you conclude if there is a solution to your SMT formula?

d. What can you conclude if there is no solution to your SMT formula?