1. (1pt) **Name:** ____

INSTRUCTIONS: Carefully read each question, and write the answer in the space provided. If answers to free response questions are written obscurely, zero credit will be awarded. The correct answer to a free response question with a short answer (i.e., one word or phrase) will never contain any significant words used in the question itself (i.e., "crossword rules"). You are permitted to use one 8.5x11 inch sheet of paper (double-sided) containing **hand-written** notes; all other aids (other than your brain) are forbidden. Questions may be brought to the instructor.

You have **150 minutes** to complete the exam.

For **TRUE** or **FALSE** and multiple choice questions, circle your answer.

On free response questions only, you will receive **20%** credit for any question which you leave blank (i.e., do not attempt to answer). Do not waste your time or mine by making up an answer if you do not know. (Note though that most questions offer partial credit, so if you know part of the answer, it is almost always better to write something rather than nothing.)

To get credit for this question, you must:

- Print your name (e.g., "Martin Kellogg") in the space provided on this page.
- Print your UCID (e.g., "mjk76") in the space at the top of **each** page of the exam.

Contents (blanks for graders only):

Writing your name and UCID:	1 / 1
I. Very Short Answer:	23 / 23
II. Matching:	20 / 20
III. Short Answer:	26 / 26
IV. DBQs:	30 / 30
V. Extra Credit:	x / 0
Total:	10x / 100

I. Multiple Choice and Very Short Answer (23pts). In the following section, either circle your answer (possible answers appear in **bold**) or write a very short (one word or one phrase) answer in the space provided. No partial credit is possible in this section.

2. (2pt) To prove a universal claim such as "these two programs produce equivalent outputs on all inputs" using an SMT solver, we need to set up the problem such that when the solver outputs **unsatisfiable**, it corresponds to the property that we want to prove.

3. (1pt) A [✓ **join**] / **meet** semi-lattice is a partially-ordered set in which each pair of elements has a unique least upper bound.

4. (2pt) The least upper bound ($\sqcup$) operator and the less-than operator ($\sqsubseteq$) in an abstract interpretation with abstract domain A must have which of these properties (select all that apply)?

   A completeness: $\forall a, b \in A. a \sqsubseteq b$ and $b \sqsubseteq a$ are defined

   [✓ **B**] completeness: $\forall a, b \in A. a \sqcup b$ is defined

   C monotonicity: $\forall a, b \in A. a \sqsubseteq a \sqcup b \wedge b \sqsubseteq a \sqcup b$

   [✓ **D**] monotonicity: $\forall a, b, c, d \in A. a \sqsubseteq b \wedge c \sqsubseteq d \rightarrow a \sqcup c \sqsubseteq b \sqcup d$

5. (2pt) When running a service like a website, you can do **A/B testing** to quantify the impact of small changes to your service on user behavior.

6. (2pt) Which of the following tools use a genetic algorithm (select all that apply)?

   [✓ **A**] EvoSuite

   B Randoop

   [✓ **C**] AFL

   D Z3

7. (1pt) [✓ **TRUE**] or **FALSE**: neither condition nor decision coverage subsumes the other.

8. (2pt) Which of these theories are typically supported by modern SMT solvers (select all that apply)?

   A boolean inequality

   B arbitrary polynomials

   [✓ **C**] equality of uninterpreted functions

   [✓ **D**] linear arithmetic

9. (1pt) **TRUE** or [✓ **FALSE**]: it is a best practice when defining the metrics in an SLA to start with the system's current performance as a baseline.

10. (2pt) Which of these is a common cause of emergencies or outages in practice (select all that apply)?

    [✓ **A**] inadequately-tested error-handling code

    B null-pointer exceptions

    [✓ **C**] configuration changes

    [✓ **D**] human or process error

11. (2pt) Iterating over a hashtable is one example of a cause of non-determinism that can lead to **flakiness** in tests: seemingly-random failures unrelated to the change being tested.

12. (2pt) Which of these techniques are commonly used to instrument a program for dynamic analysis (select all that apply)?

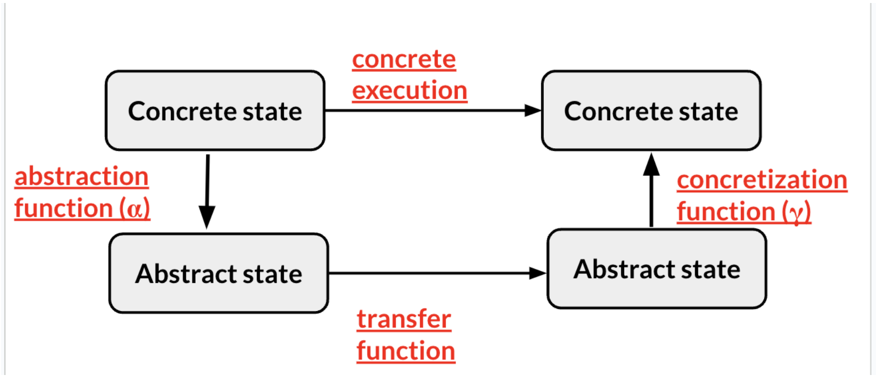
    A modified operating system (e.g., special system calls)

    B modified runtime (e.g., a special JVM)

    C modified compiler (e.g., a special C compiler)

    D automatic code rewriting (e.g., C macros)

13. (4pt) Label the four blanks in this diagram representing abstract interpretation:



A. widening	**B.** availability	**C.** modern code review	**D.** dataflow analysis
E. holistic code review	**F.** halting problem	**G.** test-driven development	**H.** abstract domain
I. property-based testing	**J.** monitoring	**K.** equivalent mutant	**L.** redundant mutant
M. satisfying assignment	**N.** staged deployment	**O.** site reliability engineer	**P.** Nelson-Oppen
Q. trusted computing base	**R.** Galois connection	**S.** microservice architecture	**T.** unit testing

II. Matching (20pts). This section contains a collection of terms discussed in class in an "Answer Bank" (choices **A.** through **T.**). Each question in this section describes a situation associated with an answer in the Answer Bank. Write the letter of the term in the Answer Bank that best describes each situation. Each answer in the Answer Bank will be used at most once.

14. (2pt) **C: modern code review** Claude gives his coworker feedback on a proposed set of changes to their software.

15. (2pt) **A: widening** Margaret wants to analyze a program using a particular lattice, but the simplest way to express that lattice has infinite height.

16. (2pt) **I: property-based testing** Instead of an oracle that specifies a concrete output value, Edsger writes a logical formula over program variables that should be true on any execution.

17. (2pt) **O: site reliability engineer** Bjarne works on a specialized team that is responsible for the operations of several services, but he also spends about half his time on writing new automation.

18. (2pt) **G: test-driven development** Before starting to fix a bug, Guido writes a test case that shows the presence of that bug.

19. (2pt) **Q: trusted computing base** Even though she has analyzed her code with a sound static analysis, Anita also needs to check something else to be 100% sure that her code is correct.

20. (2pt) **R. Galois connection** To show that his abstract interpretation is sound, Vint shows that concretizing the result of applying the transfer function to the abstraction of the original concrete state is a subset of the original concrete state.

21. (2pt) **J. monitoring** Because Niklaus is responsible for some of his service's operations, he pays special attention to code that is responsible for this, since he can't do anything about problems that he's unaware of.

22. (2pt) **L. redundant mutant** Frances computes a dynamic subsumption graph so that she directs her attention away from these.

23. (2pt) **N. staged deployment** To limit the blast radius of a potential bug, J.C.R. first applies a new code change to a small subset of his production fleet.

III. Short answer (26pts). Answer the questions in this section in at most three sentences.

24. Consider the following formula: $(a \vee \neg b \vee d) \wedge (\neg a \vee \neg b) \wedge (b \vee \neg c) \wedge \neg c \wedge (\neg a \vee \neg d \vee e) \wedge (a \vee \neg e)$.

   (a) (1pt) To apply the DPLL algorithm for finding a satisfying assignment, the input formula must have a particular structure. What is the name of that structure? **Conjunctive normal form or "CNF"**

   (b) (2pt) Is this formula in that structure already? If so, briefly explain how you know by providing a definition of the structure. If not, convert this formula into the required structure for DPLL. **It is already in CNF: a set of "ands" at the top level of the formula, with only "ors" inside each "and".**

   (c) (8pt) Carry out the DPLL algorithm to show whether or not this formula is satisfiable. Show your work by writing each intermediate formula and the reason for each simplification step. If you find a satisfying assignment, circle it. If you do not find a satisfying assignment, write and circle "unsat" once you've reached a contradiction. **2 points for a satisfying assignment, 1 point for each intermediate formula and for each justification. The answer is:**

   $(a \vee \neg b \vee d) \wedge (\neg a \vee \neg b) \wedge (b \vee \neg c) \wedge \neg c \wedge (\neg a \vee \neg d \vee e) \wedge (a \vee \neg e)$

   by unit propagation of $\neg c$:

   $(a \vee \neg b \vee d) \wedge (\neg a \vee \neg b) \wedge (\neg a \vee \neg d \vee e) \wedge (a \vee \neg e)$

   by pure literal elimination of $\neg b$: $(\neg a \vee \neg d \vee e) \wedge (a \vee \neg e)$

   by pure literal elimination of $\neg d$:

   $(a \vee \neg e)$

   This is clearly satisfiable. A satisfying assignment is $a, \neg b, \neg c, \neg d, \neg e$.

25. (6pt) Consider an abstract interpretation that uses the lattice on the left-hand side below, and whose abstract values have the (standard) meanings defined on the right-hand side below.

$$\begin{tikzcd}
 & \top & \\
\text{Odd} \arrow[ur] & & \text{Even} \arrow[ul] \\
1 \arrow[u] & & \text{Even} \neq 0 \arrow[u] \\
 & \bot \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

$\top$	:	$\mathbb{Z}$	Even	:	$\{x.x\%2 = 0\}$
1	:	$\{1\}$	Odd	:	$\{x.x\%2 = 1\}$
$\bot$	:	$\{\}$	Even $\neq 0$	:	$\{x.x\%2 = 0 \wedge x \neq 0\}$

Recall that the factorial $n!$ of some integer n is defined as the product of all positive integers less than or equal to n. Fill out the following transfer function for the factorial operator (!) for the above abstract interpretation as precisely as you can:

$\top$	->	$\top$
Even	->	$\top$
Odd	->	$\top$
1	->	**1**
Even $\neq 0$	->	**Even**$\neq 0$
$\bot$	->	$\bot$

1 point each.

26. (4pt) Suppose that you are a software engineer at Black Rook, a hedge fund whose founder loved chess. To test that the software you're working on won't crash under real-world conditions, before the software is released, your manager directs you to choose a large number of random inputs and make sure that your code doesn't crash when run on them. Support or refute the following claim: this activity is toil. **Likely support. Though this activity does have long-term enduring value, it is entirely automatable. In particular, full credit answers must mention fuzzing as an alternative tool that accomplishes the same thing and is readily available.**

27. Recall that DPLL(T) cannot use one of the heuristics that "standard" DPLL can use.

   (a) (2pt) Which heuristic can DPLL(T) not use? **Pure literal elimination.**

   (b) (3pt) Briefly explain why DPLL(T) cannot use this heuristic. **Because each variable in a DPLL(T) formula may represent a theory clause, and the theory clauses may not be independent. So, just because a variable only appears positively or negatively does not mean that setting it to true or false, respectively, cannot hurt the solver.**

IV. Document-based Questions (30pts). All questions in this section refer to a documents **A-C**. These documents appear at the end of the exam (I recommend that you tear them out and refer to them as you answer the questions).

Questions on this page refer to **Documents A** and **B**, which each contain a proposed lattice for an abstract interpretation about the signs of integers. Such an abstract interpretation might be useful for proving that arrays are created with non-negative sizes, among other tasks.

28. (6pt) Give a program that is safe, but for which abstract interpretations based on these two lattices would give different results in such a way that one of them would produce a false positive warning about a possible negative array creation, but the other would prove that the program is safe. **The simplest example involves a non-negative array creation, which the document A lattice will treat as top. For example, consider:**

```
if (*):
   x = 0
else:
   x = 1
a = new int[x]
```

29. (6pt) Give a (safe) program for which abstract interpretations based on both lattices will produce a false positive warnings. **The easiest way to answer this question and the next is to exploit the fact that there is no Zero abstract value in Document B. For example, both lattices can't prove the program below is safe, but adding a Zero AV to Document B allows them to do so:**

```
x = -1
y = 0
if (*):
  z = x * y
else:
  z = 1
a = new int[z]
```

30. (4pt) Suggest a new lattice based on either **Document A** or **B** that would permit an abstract interpretation to prove that your answer to question 29 is safe. Your new lattice may contain no more than one new abstract value than the lattice on which it is based. **Answers vary, and carried-thru-errors will be accounted for. See the answer to question 29 for the simplest answer.**

Questions on this page refer to **Document C**. Assume that the only statements in the program are `STATEMENT 1` through `STATEMENT 4`, which are the statements inside the conditionals.

31. (a) (4pt) Write the path predicate for each statement.

    STATEMENT 1: $x > y$

    STATEMENT 2: $x \leq y \wedge (x < 4 \vee y > 7)$

    STATEMENT 3: $x \leq y \wedge x \geq 4 \wedge y \leq 7 \wedge y < 3$

    STATEMENT 4: $x \leq y \wedge x \geq 4 \wedge y \leq 7 \wedge y \geq 3$

    (b) (1pt) What is the highest possible statement coverage that a test suite can achieve on this program?

    75%

    (c) (2pt) Write a test suite for `foo` that achieves the statement coverage in your answer to question 31b. Express your answer as a list of tuples, e.g., "(x = #, y = #), (x = #, y = #), etc.", where each "#" is a specific integer.

    (x = 6, y = 0), (x = 3, y = 8), (x = 4, y = 5)

32. Consider the mutation operator "< to <=".

    (a) (1pt) How many first-order mutants can be produced by applying this mutation operator to `foo`?

    2

    (b) (2pt) What is the mutation score of a test suite containing only the following test input (assume that executing a different statement causes the mutant to be killed) on the mutants created by this mutation operator: (x = 5, y = 4)?

    0%

    (c) (2pt) Is it possible to change one of x or y in the test input in question 32b to improve the mutation score? If so, state which of x or y to change, what its new value should be, and what the new mutation score is. If not, explain why not.

    x = 4 changes the mutation score to 50%

    (d) (2pt) Is it possible to change one of x or y in your answer to question 32c to further improve the mutation score? If so, state which of x or y to change, what its new value should be, and what the new mutation score is. If not, explain why not.

    It is not, because the second mutant is an equivalent mutant: STATEMENT 3 is still unreachable.

V. Extra Credit. Questions in this section do not count towards the denominator of the exam score.

33. (1pt) In section III (Matching), there is a theme to the names used in the situation descriptions. What is the theme? **First names of famous computer scientists: Claude Shannon, Edsger Dijkstra, Margaret Hamilton, Bjarne Stroustrup, Guido van Rossum, Anita Borg, Vint Cerf, Niklaus Wirth, Frances Allen, J.C.R. Licklider.**

34. (1pt) Give the last name of one of the people whose first name was used in section III (Matching) and describe what they did that makes them fit the theme. **Any correct answer with any interesting fact about one of the famous computer scientists gets the point.**

35. (1pt) Name an SMT solver other than Z3. **cvc4, cvc5, and many other valid answers.**

36. (1pt) Who is credited with inventing abstract interpretation? **Patrick and Radhia Cousot. Just "Cousot" also gets the point.**

37. (1pt) Explain something interesting that you learned in this class since the midterm, but that this exam didn't test. **Any reasonable answer gets the point.**

This page intentionally left blank (you may use it as scratch paper, and the proctor will have more scratch paper at the front if you need more).

Document A:

$$\begin{tikzcd}
 & \top & \\
\text{Positive } (> 0) \arrow[ur] & \text{Zero } (= 0) \arrow[u] & \text{Negative } (< 0) \arrow[ul] \\
 & \bot \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Document B:

$$\begin{tikzcd}
 & \top & \\
\text{Non-negative } (\geq 0) \arrow[ur] & & \text{Non-positive } (\leq 0) \arrow[ul] \\
\text{Positive } (> 0) \arrow[u] & & \text{Negative } (< 0) \arrow[u] \\
 & \bot \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Document C

```
void foo(int x, int y) {
  if (x > y) {
    // STATEMENT 1
  } else if (x < 4 || y > 7) {
    // STATEMENT 2
  } else if (y < 3) {
    // STATEMENT 3
  } else {
    // STATEMENT 4
  }
}
```