THE UNIVERSITY OF QUEENSLAND AUSTRALIA

This exam paper must not be removed from the venue

Venue	____________________
Seat Number	________
Student Number	|__|__|__|__|__|__|__|__|__|
Family Name	______________________
First Name	______________________

School of Information Technology and Electrical Engineering

EXAMINATION

Semester Two Final Examinations, 2021

CSSE4630 Principles of Program Analysis

This paper is for St Lucia Campus students.

Examination Duration:	120 minutes
Reading Time:	10 minutes

Exam Conditions:

This is a Closed Book examination - specified written materials permitted

Any calculator permitted - unrestricted

During reading time (= planning time) - students are encouraged to review and plan responses to the exam questions

This examination paper will be released to the Library

Materials Permitted In The Exam Venue:

(No electronic aids are permitted e.g. laptops, phones)

Blank scrap paper permitted - unlimited sheets permitted

One A4 sheet of handwritten or typed notes double sided is permitted

Materials To Be Supplied To Students:

1 x 14-Page Answer Booklet

Instructions To Students:

Additional exam materials (eg. answer booklets, rough paper) will be provided upon request.

Attempt all questions (total possible is 100 marks).

The value of each question is indicated on the question.

For Examiner Use Only

Question	Mark

Total ________

1. **[25 marks total]** Short Answer questions

   (a) **[3 marks]** Calculate the solution of unifying the following two type terms, where S, T are type variables, a, b are constants and d is a data type constructor. If the two terms are not unifiable, explain why.
   $d(a, S) = d(T, d(T, b))$

   (b) **[6 marks]** For a live variables static analysis of a program with variables $\{a, b, c, d\}$, we use the REVERSE powerset lattice L, ordered by $\supseteq$. Given $S = \{a, b\}$ and $T = \{b, c\}$, calculate each of the following lattice expressions:

   - $\bot$
   - $S \sqcap T$
   - $S \sqcup T$

   (c) **[8 marks]** For a static analysis of a program with integer constants $B = \{-\inf, 0, 1, 10, 100, +\inf\}$ we use the interval lattice L, where for any interval (lo, hi), we usually have lo and hi being integers with $lo \leq hi$, but lo may be negative infinity $(-\inf)$ and hi may be positive infinity $(+\inf)$. The lattice is ordered by interval inclusion, with the empty interval at the bottom and the full interval $(-\inf, +\inf)$ at the top. To ensure termination of our analysis, we also use the interval widening operator $\nabla' : L \times L \to L$, that takes two intervals and returns a possibly-widened interval. Calculate each of the following interval expressions:

   - $[1, 3] \sqcap [2, 5]$
   - $[1, 3] \sqcup [2, 5]$
   - $[2, 9]\nabla'[1, 9]$
   - $[1, 9]\nabla'[0, 11]$

   (d) **[2 marks]** Give an example of a static analysis that uses a Forward-MUST analysis.

   (e) **[2 marks]** Give an example of a static analysis that uses a Backward-MAY analysis.

   (f) **[4 marks]** Calculate the least fixed point solution of the following functions of type $L \to L$ where L is the powerset lattice $(\{a, b, c, d\}, \subseteq)$.

   - $f(s) = s \cup \{b, c\}$
   - $g(s) = s \cap \{a, b, c\}$

2. **[10 marks total]** Define static analysis and dynamic analysis of programs, and compare the advantages and disadvantages of the two approaches.

3. **[20 marks total]** We want to implement a static analysis for propagating constant integer values through programs.

   We choose as our abstract value lattice L the flat lattice of integers ($\mathbb{Z}$). At each control-flow-graph (CFG) node our analysis state is $Vars \to L$, where $Vars$ is the set of program variables at that point. For each integer operator op we define an abstract operator $\hat{op}$ such that:

$$a \; \hat{op} \; b = \begin{cases} \bot, & if\ a = \bot \vee b = \bot \\ \top, & otherwise,\ if\ a = \top \vee b = \top \\ a\ op\ b, & if\ a \in \mathbb{Z}\ \wedge\ b \in \mathbb{Z} \end{cases}$$

   We define an expression evaluation function $eval(s, E)$ as:

$$\begin{aligned} eval(s, X) &= s(X) \\ eval(s, I) &= I, \qquad \text{for any integer constant } I \\ eval(s, \text{input}) &= \top \\ eval(s, E_1\ op\ E_2) &= \hat{op}(eval(s, E_1), eval(s, E_2)) \end{aligned}$$

   Here are the constraint rules for this static analysis. For a control-flow-graph (CFG) node v of the form:

$$\begin{aligned} [\![X = E]\!] &= JOIN(v)[X \mapsto eval(JOIN(v), E)] \\ [\![var\ X_1, \ldots, X_n]\!] &= JOIN(v)[X_1 \mapsto \bot, \ldots, X_n \mapsto \bot] \end{aligned}$$

and for all other CFG nodes we have just $[\![v]\!] = \mathit{JOIN}(v)$. Note that $\mathit{JOIN}(v) = \bigsqcup_{w \in pred(v)} [\![w]\!]$, where $pred(v)$ is the set of predecessor nodes of v in the CFG.

We will use these rules to analyse the following simple TIP function.

```
 1: f(x) {
 2:     var y,z;
 3:     y = 100;
 4:     if (x > 0) {
 5:         z = x * y;
 6:         output(z);
 7:     } else {
 8:         x = 1000;
 9:         z = x / 2;
10:         output(z);
11:     }
12:     return x + z;
13: }
```

(a) **[6 marks]** Write all the constraints generated for each statement in the function.

(b) **[4 marks]** Analyse the function f starting with the ENTRY state $\{x \mapsto \top\}$. Solve your constraints and write down the state of your analysis after lines 5, 9 and 11.

(c) **[4 marks]** Now we will perform an interprocedural analysis of the function call $f(3)$, using the functional approach to context sensitivity. Analyse the function by solving your constraints again, starting with the ENTRY state $\{x \mapsto 3\}$. Write down the state of your analysis after lines 5, 9 and 11.

(d) **[6 marks]** Explain the advantages of doing a constant propagation analysis such as this, and the kinds of compiler optimisations that it can enable. Use your analysis results of the function f to illustrate your points if appropriate.

4. **[25 marks total]** We want to perform a static *points-to* analysis of the following TIP program, to determine possible aliasing between variables, in order to generate optimised code in a compiler.

```
 1: main() {
 2:   var a,b,c,d;
 3:   a = alloc 5;
 4:   b = alloc 7;
 5:   c = alloc 9;
 6:   *a = b;
 7:   *b = c;
 8:   // search down list
 9:   d = b;
10:   if (d) {
11:     d = *d;
12:   }
13:  return 0;
14: }
```

(a) **[10 marks]** Use Andersen's algorithm to perform the points-to analysis. The constraint rules for the Andersen static analysis are as follows:

$$\begin{array}{rl}
[\![X = \mathit{alloc}\ P]\!]: & \text{alloc-}i \in [\![X]\!] \\
[\![X_1 = \&X_2]\!]: & X_2 \in [\![X_1]\!] \\
[\![X_1 = X_2]\!]: & [\![X_2]\!] \subseteq [\![X_1]\!] \\
[\![X_1 = *X_2]\!]: & c \in [\![X_2]\!] \Longrightarrow [\![c]\!] \subseteq [\![X_1]\!] \text{ for each } c \in \mathit{Cells} \\
[\![*X_1 = X_2]\!]: & c \in [\![X_1]\!] \Longrightarrow [\![X_2]\!] \subseteq [\![c]\!] \text{ for each } c \in \mathit{Cells}
\end{array}$$

i. Write all the constraints generated from the above program, using these Andersen rules.

ii. Solve your constraints to find the minimal solution.
iii. Draw the points-to graph that corresponds to your solution.

(b) **[10 marks]** Use Steensgaard's algorithm to perform the points-to analysis. The unification-based constraint rules for the Steensgaard static analysis are as follows:

$$\begin{aligned}
[\![X = alloc\ P]\!] &: \quad [\![X]\!] = \uparrow \text{alloc-}i \\
[\![X_1 = \&X_2]\!] &: \quad X_1 = \uparrow [\![X_2]\!] \\
[\![X_1 = X_2]\!] &: \quad [\![X_1]\!] = [\![X_2]\!] \\
[\![X_1 = *X_2]\!] &: \quad [\![X_1]\!] = \alpha \wedge [\![X_2]\!] = \uparrow \alpha \quad \text{for fresh } \alpha \\
[\![*X_1 = X_2]\!] &: \quad [\![X_1]\!] = \uparrow \alpha \wedge [\![X_2]\!] = \alpha \text{ for fresh } \alpha
\end{aligned}$$

i. Write all the constraints generated from the above program, using these Steensgaard rules.
ii. Solve your unification constraints to find the minimal solution.
iii. Draw the points-to graph that corresponds to your solution.

(c) **[5 marks]** Compare and contrast the Andersen and Steensgaard algorithms, using your above solutions to illustrate your points.

5. **[20 marks total]** Recall that concolic testing uses a mixture of concrete and symbolic execution of a program.

(a) **[8 marks]** Explain the goal of concolic testing, and give a brief overview of how it works (2-3 sentences).

(b) **[12 marks]** Use concolic testing to test the following TIP function, starting with inputs $x = 0, y = 0$. Write down the sequence (or tree) of test cases that it would find. State clearly each time it invokes the SMT solver, showing the inputs to the SMT solver, and the outputs that it returns.

```
testme(x, y) {
  var z;
  z = 1;
  if (1 > y) {
    error 1;
  }
  while (x > y) {
    if (x > 2*y) {
      error 2;
    }
    z = z * x;
    x = x - 1;
  }
  return z;
}
```

END OF EXAMINATION