

Final Exam

EDAP15: Program Analysis, HT 2022

2022-01-12

Anonymisation code: _____

PLEASE READ THE FOLLOWING CAREFULLY

This final exam consists of 10 questions. You can reach a total of 100 points. If you get 50 points (including your bonus points from the homework assignments), you will pass the exam.

Make sure that the final exam consists of precisely 19 numbered pages. Write down your anonymisation code on each page. If you have no anonymisation code, write down your personnummer/personal code.

Do not write in any light or red colour.

Only supply one answer per question. Strike out incorrect answers.

If you run out of space, continue writing on the back of the page. Additional sheets are available.

The following utilities are permitted:

- Paper and writing material
- Calculators that are not capable of wireless connectivity (should not be necessary)
- Two sheets of A4 paper with hand-written notes (possibly on both sides)

Make sure to read all questions carefully before starting on your answer!

Good luck!

Question:	1	2	3	4	5	6	7	8	9	10	Sum
Max Points:	7	11	14	9	6	11	8	7	10	17	100
Points Reached:											

Question 1 (7 Points)

Consider the following Java code:

```
static int h100(x : int) {
    int x = 1
    int y = 0;
    int t = 0;

    while (x > 1) {
        if (0 == x % 2) {
            x = x / 2;
        } else {
            x = x * 3 + 1;

            if (x > 100) {
                t += 1;
            }
        }

        y += 1;
    }
    return t;
}
```

```
static String v;

static void f(int z) {
    if (z == 0) {
        v = null;
        g();
    } else {
        v = "up-";
        g();
    }
    print(v);
}

static void g() {
    if (v == null) {
        return;
    }
    v = v + "down";
}
```

- (a) (3 Points) Draw an *intraprocedural* Control-Flow Graph (CFG) that accurately reflects the body of `h100`.

(b) (4 Points) Draw an *interprocedural* CFG that accurately reflects functions `f` and `g`.

```
static void f(int z) {  
    if (z == 0) {  
        v = null;  
  
        g();  
    } else {  
        v = "up-";  
  
        g();  
    }  
    print(v);  
}  
  
static String v;  
  
static void g() {  
    if (v == null) {  
        return;  
    }  
    v = v + "down";  
}
```

Question 2 (11 Points)

Consider the following program in Java:

```
1 static void main(String[] args) {
2     N a = new N(null);
3     N b = new N(a);
4     N c = new N(b);
5     // [A]
6     c = alloc(alloc(alloc(c)));
7     b = c.x.x;
8     // [B]
9 }

10 static N alloc(N v) {
11     return new N(v);
12 }
13
14 class N {
15     public N(Object vx) {
16         this.x = vx;
17     }
18     public Object x;
19 }
```

For simplicity, assume that `args = null`.

(a) (2 Points) Draw the concrete heap graph at the point marked with [A]. Omit edges to `null`.

(b) (3 Points) Draw the concrete heap graph at the point marked with [B]. Omit edges to `null`.

- (c) (4 Points) Draw an abstract heap graph for location [B] with *allocation-site based summaries*. Omit any edges to `null`.
- (d) (2 Points) Assume that we are using the abstract heap graph to determine *aliasing*. Are there any cases in which the abstract heap graph and the concrete heap graph will disagree about whether two access paths may alias each other? If there is a difference, give an example, otherwise explain why there is no difference.

Question 3 (14 Points)

We want to analyse the following language with *monomorphic type inference*:

$$\begin{array}{l}
 \langle Prog \rangle \longrightarrow \langle Stmt \rangle \mid \langle Stmt \rangle \langle Prog \rangle \\
 \langle Stmt \rangle \longrightarrow \langle Var \rangle := \langle Expr \rangle ; \\
 \langle Expr \rangle \longrightarrow \text{empty} \\
 \quad \mid \{ \langle Expr \rangle \} \\
 \quad \mid \mathbf{0} \mid \mathbf{1} \\
 \quad \mid \langle Expr \rangle + \langle Expr \rangle \\
 \quad \mid \langle Expr \rangle - \langle Expr \rangle \\
 \quad \mid @ \langle Expr \rangle \\
 \quad \mid \langle Var \rangle \\
 \langle Var \rangle \longrightarrow \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \dots
 \end{array}
 \qquad
 \begin{array}{l}
 \mathbf{Types:} \\
 \langle Type \rangle \longrightarrow \text{INT} \\
 \quad \mid \text{SET}[\langle Type \rangle]
 \end{array}$$

We use the following typing rules:

$$\begin{array}{c}
 \frac{e_1 : \text{INT} \quad e_2 : \text{INT}}{e_1 + e_2 : \text{INT}} \text{ (addi)} \qquad \frac{e_1 : \text{SET}[\tau] \quad e_2 : \text{SET}[\tau]}{e_1 + e_2 : \text{SET}[\tau]} \text{ (adds)} \qquad \frac{e_1 : \text{SET}[\tau] \quad e_2 : \text{SET}[\tau]}{e_1 - e_2 : \text{SET}[\tau]} \text{ (subs)} \\
 \\
 \frac{}{\mathbf{0} : \text{INT}} \text{ (zero)} \qquad \frac{}{\mathbf{1} : \text{INT}} \text{ (one)} \qquad \frac{e : \tau}{\{e\} : \text{SET}[\tau]} \text{ (single)} \qquad \frac{e : \text{SET}[\tau]}{@e : \tau} \text{ (choice)} \\
 \\
 \frac{v \in \text{Var} \quad \Delta(v) = \bar{\tau}}{v : \bar{\tau}} \text{ (var)} \qquad \frac{\Delta(v) = \bar{\tau} \quad e : \bar{\tau}}{v := e ; : \bar{\tau}} \text{ (assign)}
 \end{array}$$

As in the lectures and analogously to homework assignment 1, we assume that:

- Each variable is uniquely identified by its name (i.e., there is only one global scope)
- We use notation $\Delta(v) = \bar{\tau}$ to mark that variable v must have type $\bar{\tau}$
- We require that a well-formed program has a type for all statements (per the *(assign)* rule)

(a) (7 Points) Consider the following program:

```

x := y;
y := { 0 };
a := y - @z;

```

Assume that you have implemented a type inference system analogous to the one from the first homework exercise. On the table on the next page, write all type constraints that you would generate for the program above; we have already filled in the first row to illustrate the format. If you introduce new type variables, use the names α, β, γ , possibly with indices (α_1 etc.). Also write down which rules you used to derive each constraint. (You do not have to write down where you used the *(var)* rule.)

Note: there are several correct solutions, with varying numbers of constraints.

Question 4 (9 Points)

Andersen's Points-To Analysis is one of the most important algorithms for the analysis of programs with reference semantics. For the questions below, assume that we are using the *field-insensitive* version of the algorithm, as in the corresponding homework assignment.

(a) (1 Point) Andersen's algorithm is *graph-based*. What do the *vertices* in Andersen's graph represent?

(b) (4 Points) What do the edges in Andersen's graph represent? Explain with one example per type of edge.

(c) (4 Points) Consider the following code in Teal:

```
a.x := b.x;
```

What vertices and edges would represent this code in Andersen's graph? Keep the set of edges minimal, but otherwise ensure sound over-approximation (i.e., your solution must not induce false negatives in the generated points-to sets).

Question 5 (6 Points)

We are analysing a Teal-like language and want to find all variables that are *tainted* in the sense that they may contain the result of a call to a special built-in function `unsafe()`, at any given program point:

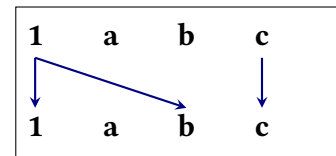
```
a := 0;
    // a is not tainted
a := unsafe();
    // a is tainted
a := 1;
    // a is no longer tainted
```

- (a) (1 Point) Assume that we are performing this analysis to detect security flaws in an online banking system. Would you use a MUST or a MAY analysis? Justify your answer.

- (b) (5 Points) We want to perform the analysis using the IFDS algorithm. IFDS uses *representation relations* to encode its results, so that it might e.g. encode the block

```
// variables a, b, c
a := 0;
b := unsafe();
// c unchanged
```

as representation relation:



What representation relation would you compute for the following code:

```
// variables x, y, z
x := unsafe();
if z == y {
    x := 0;
    y := unsafe();
} else {
    z := y;
}
```

Make sure that your answer is consistent with your answer to sub-question (a). Hint: write down intermediate results.

Question 6 (11 Points)

We are performing interval analysis on a Teal-like language with the usual arithmetic operations, to determine the possible ranges of numbers that variables may take at different points in the code. We are implementing this analysis using data flow analysis on the *Interval Domain*.

Assume for simplicity that the language semantics only uses “big integers”, i.e., integers that can grow arbitrarily large or small.

- (a) (5 Points) Write down maximally precise transfer functions $trans(E)$, where E is a partial map that represents the product lattice. There are four cases below, the first of which is already filled in; fill in the remaining ones:

Source code	Transfer Function
$x := y$	$trans(E) = E[x \mapsto E(y)]$
$x := 0$	$trans(E) = E[x \mapsto \quad]$
$x := y + z$	$trans(E) = E[x \mapsto \quad]$
$x := abs(y)$	$trans(E) = E[x \mapsto \quad]$

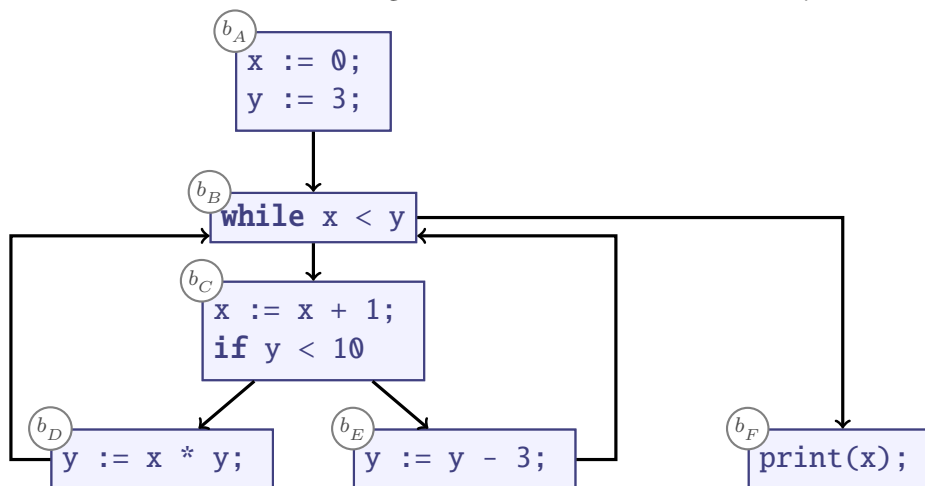
Here, $abs(y)$ has the usual meaning of $|y| = \begin{cases} y & \iff y \geq 0 \\ -y & \iff y \leq 0 \end{cases}$

- You may write the transfer functions formally and/or in English, as you prefer, but be precise.
 - If there are multiple cases, make sure to cover all of them.
 - To “unpack” an interval, you can write e.g. “where $E(y) = [y_l, y_r]$ ” and then use y_l and y_r in the body of the transfer function.
- (b) (2 Points) Assume that we are performing data flow analysis on the interval domain, using maximally precise transfer functions like yours. Is the data flow analysis guaranteed to terminate?

Answer ONLY ONE of sub-questions (c) and (d), as determined by your answer.

- (c) **Only if you answered “no, the analysis is *not* guaranteed to terminate”** in sub-question (b): Explain how we can modify the analysis to terminate without making it completely useless. Illustrate with a short example. (4 points)

- (d) **Only if you answered “yes, the analysis *is* guaranteed to terminate”** in sub-question (b): Assume that we are applying the analysis on the CFG below. For each CFG node, write down the lattice element assigned to variable x after the analysis terminates. (4 points)



Question 7 (8 Points)

- (a) (3 Points) You are attempting to analyse the possible types that different variables in your program can take. Give two arguments in favour of using a *dynamic* analysis for this task.
- (b) (3 Points) Give two arguments in favour of using a *static* analysis for the same task.
- (c) (2 Points) Some dynamic analyses are affected by *perturbation*. Is perturbation a concern for such a dynamic type analysis? Explain.

Question 8 (7 Points)

You are building an analysis to detect the length of arrays in Teal-0. The analysis is specifically designed to support a new function

$$\text{concat} : \text{ARRAY}[\tau] \times \text{ARRAY}[\tau] \rightarrow \text{ARRAY}[\tau]$$

that concatenates two arrays, i.e., $\text{concat}(\mathbf{a}, \mathbf{b})$ returns a new array that contains all the elements of \mathbf{a} and \mathbf{b} , in order.

The analysis is as follows:

- The **per-variable lattice** is $\mathbb{Z}_{\perp}^{\top}$, the integers lifted to include a distinguished top element and a distinguished bottom element.
- The main **transfer functions** are as follows:

Source	Transfer Function
$x := y$	$\text{trans}(E) = E[x \mapsto E(y)]$
$x := \text{new array}[\tau](n)$	$\text{trans}(E) = E[x \mapsto n]$
$x := [e_1, \dots, e_n]$	$\text{trans}(E) = E[x \mapsto n]$
$x := \text{concat}(y, z)$	$\text{trans}(E) = E[x \mapsto E(y) +_{\perp} E(z)]$

where x, y, z are variables, τ are types, and n are integers, and we define

$$a +_{\perp} b = \begin{cases} \top & \iff a = \top \text{ or } b = \top \\ a + b & \iff a \in \mathbb{Z} \text{ and } b \in \mathbb{Z} \\ \perp & \text{otherwise (one of } a, b \text{ is } \perp \text{ and neither is } \top) \end{cases}$$

(a) Is this a *Monotone Framework*? Explain your reasoning.

(b) Is this a *Distributive Framework*? Explain your reasoning.

Question 9 (10 Points)

You are asked to build a program analysis to support software for a satellite. The satellite has four *thrusters*, little jets that it can switch on or off to control its rotation and position. However, it may at any given time only have two thrusters switched on, to avoid overheating. The satellite control software is (inexplicably) written in Teal, which has three special new built-in operations to control the thrusters:

`thruster_on` : $\text{INT} \rightarrow \text{INT}$ `thruster_on(t)` switches on thruster t and returns 1.
`thruster_off` : $\text{INT} \rightarrow \text{INT}$ `thruster_off(t)` switches off thruster t and returns 0.
`thruster_active`: $\text{INT} \rightarrow \text{INT}$ `thruster_active(t)` returns 1 if thruster t is on, otherwise 0 (thruster is off).

For all three operations, the thruster number t must be $0 \leq t \leq 3$.

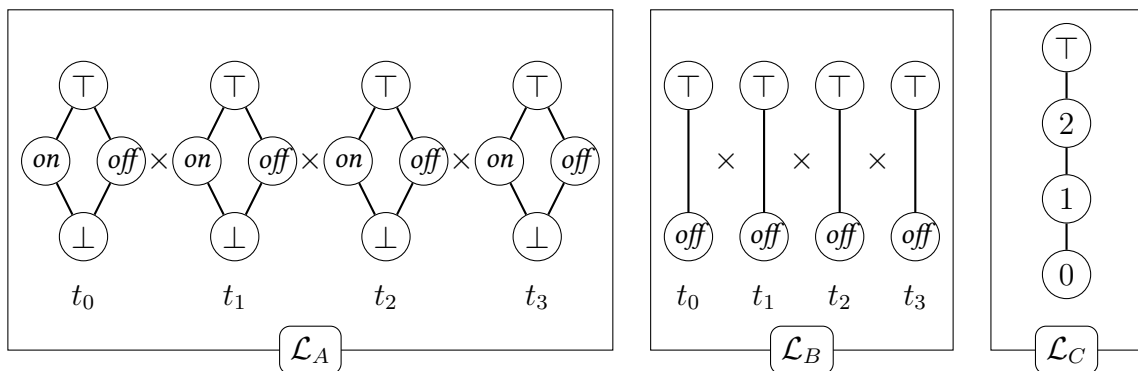
Your manager splits up the program analysis challenges into two parts:

- **Thruster Oversupply Analysis:** Detects any program points at which more than two thrusters may be active.
- **Thruster Redundant Switch Analysis:** Detects any `switch_on` that is called on a thruster that is already on, and any `switch_off` that is called on a thruster that is already off.

The satellite control software team will treat any reports by either analysis as critical bugs in the software.

Your task is to build the **Thruster Oversupply Analysis**. For now, assume that the code only uses the three thruster operations with literal numbers as parameters (e.g., `thruster_on(1)`, but never `thruster_on(x)` or `thruster_on(2+3)`).

- (a) (2 Points) After consulting with an expert for satellite software verification, your manager proposes the following three analysis lattices to you:



- \mathcal{L}_A and \mathcal{L}_B are product lattices that separately reflect the state for thrusters $t_0 \dots t_3$.
- \mathcal{L}_A tracks whether the thruster is *on*, *off*, or in an unknown state (\top , \perp).
- \mathcal{L}_B tracks whether the thruster is definitely *off*, or in an unknown state (\top).
- \mathcal{L}_C tracks the maximum number of thrusters that might currently be active.

Pick one of the lattices. Any choice is valid. Justify with a technical argument (e.g., analysis quality, software engineering quality) why you prefer your lattice over the other two.

Use the same lattice for the remaining sub-questions.

(b) (4 Points) Describe the transfer functions for `thruster_on` and `thruster_off` for your choice of lattice. You can use the notation used earlier or explain their behaviour in English.

(c) (4 Points) Another team has developed an interval analysis for `INT` values. Their interval analysis models the three thruster operations conservatively and assumes that `thruster_active` may always return either 0 or 1.

Your manager proposes the following ideas:

- Option (CT): Your analysis could be an analysis client of the interval analysis.
- Option (CI): The interval analysis could be a client of your analysis.

For both options, explain what potential benefits you see, if any. If you see no benefits, explain why.

(e) (3 Points) What is the purpose of a call graph, i.e., when and how can it be useful? Explain with an example.

(f) (2 Points) Some analyses are *flow sensitive*. When is this property important for an analysis? Explain with an example.

(g) (2 Points) When is flow-sensitivity *not* important for an analysis? Explain with an example.

(h) (4 Points) Why is it beneficial for some analyses if they are *call-site sensitive*? Give an example to demonstrate what call-site sensitivity means and how it can be beneficial.

Anonymkod: