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 16 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)

Possible solution or hints: This version includes hint/solution boxes, marked in this colour. The solutions are not always complete, but highlight the main points to consider.

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;
}
```

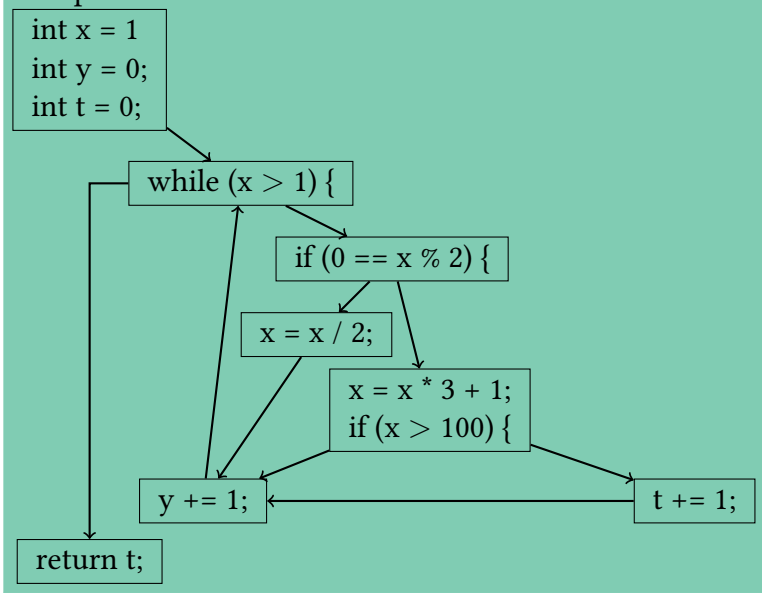
```
static String v;

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 *intra*procedural Control-Flow Graph (CFG) that accurately reflects the body of `h100`.

One possible solution with Basic Blocks:



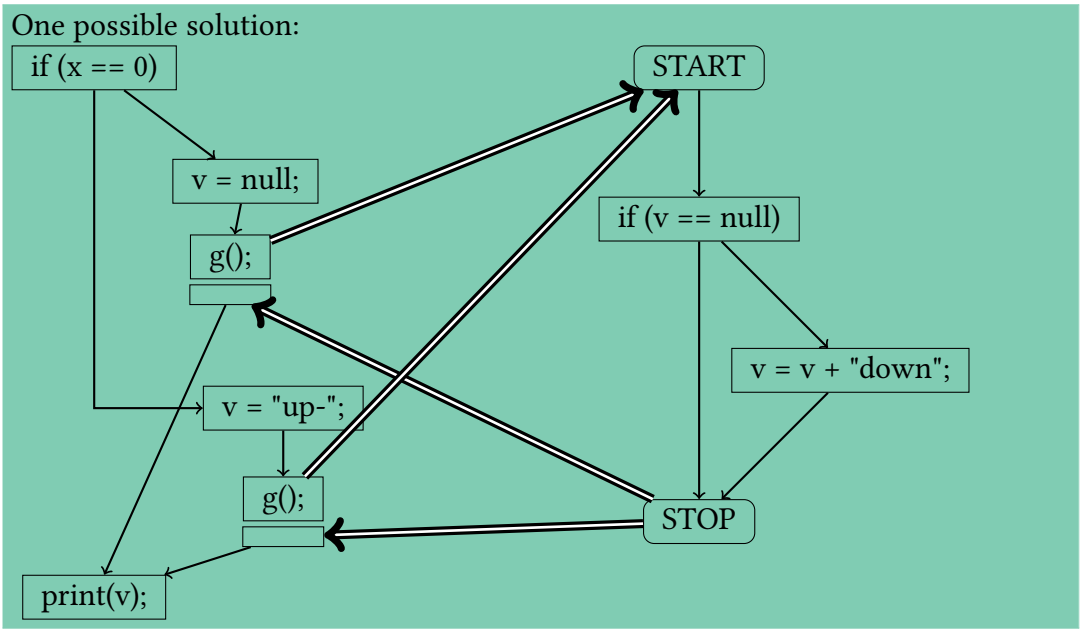
(b) (4 Points) Draw an *inter*procedural CFG that accurately reflects functions `f` and `g`.

```
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";
}
```

One possible solution:



Question 2 (11 Points)

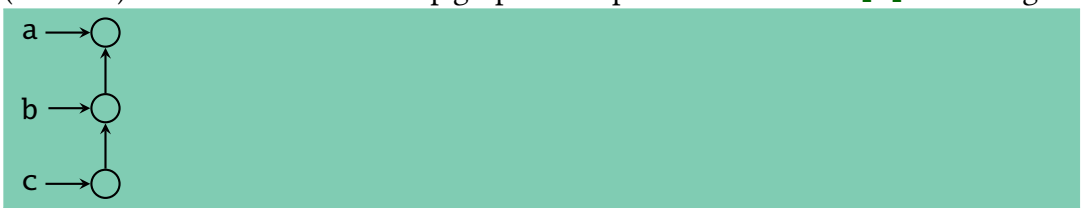
Consider the following program in Java:

```
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.

The AHG will consider a and b to possibly be aliased, since both point to the summary node that represents allocation site 11.

Question 3 (14 Points)

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

$$
\begin{array}{lcl}
\langle \mathit{Prog} \rangle & \longrightarrow & \langle \mathit{Stmt} \rangle \mid \langle \mathit{Stmt} \rangle\ \langle \mathit{Prog} \rangle \\
\langle \mathit{Stmt} \rangle & \longrightarrow & \langle \mathit{Var} \rangle\ \texttt{:=}\ \langle \mathit{Expr} \rangle\ \texttt{;} \\
\langle \mathit{Expr} \rangle & \longrightarrow & \texttt{empty} \\
& \mid & \texttt{\{}\langle \mathit{Expr} \rangle\texttt{\}} \\
& \mid & \texttt{0} \mid \texttt{1} \\
& \mid & \langle \mathit{Expr} \rangle\ \texttt{+}\ \langle \mathit{Expr} \rangle \\
& \mid & \langle \mathit{Expr} \rangle\ \texttt{-}\ \langle \mathit{Expr} \rangle \\
& \mid & \texttt{@}\ \langle \mathit{Expr} \rangle \\
& \mid & \langle \mathit{Var} \rangle \\
\langle \mathit{Var} \rangle & \longrightarrow & \texttt{a} \mid \texttt{b} \mid \texttt{c} \ldots
\end{array}
$$

Types:

$$
\begin{array}{lcl}
\langle \mathit{Type} \rangle & \longrightarrow & \textsc{Int} \\
& \mid & \textsc{Set}[\langle \mathit{Type} \rangle]
\end{array}
$$

We use the following typing rules:

$$\frac{e_1 : \textsc{Int} \quad e_2 : \textsc{Int}}{e_1\ \texttt{+}\ e_2 : \textsc{Int}}\ (\mathit{addi}) \qquad \frac{e_1 : \textsc{Set}[\tau] \quad e_2 : \textsc{Set}[\tau]}{e_1\ \texttt{+}\ e_2 : \textsc{Set}[\tau]}\ (\mathit{adds}) \qquad \frac{e_1 : \textsc{Set}[\tau] \quad e_2 : \textsc{Set}[\tau]}{e_1\ \texttt{-}\ e_2 : \textsc{Set}[\tau]}\ (\mathit{subs})$$

$$\frac{}{\texttt{0} : \textsc{Int}}\ (\mathit{zero}) \qquad \frac{}{\texttt{1} : \textsc{Int}}\ (\mathit{one}) \qquad \frac{e : \tau}{\texttt{\{}e\texttt{\}} : \textsc{Set}[\tau]}\ (\mathit{single}) \qquad \frac{e : \textsc{Set}[\tau]}{\texttt{@}e : \tau}\ (\mathit{choice})$$

$$\frac{v \in \mathit{Var} \quad \Delta(v) = \tau}{v : \tau}\ (\mathit{var}) \qquad \frac{\Delta(v) = \tau \quad e : \tau}{v\texttt{:=}e\texttt{;} : \tau}\ (\mathit{assign})$$

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 $\Delta(v) = \tau$ to mark that variable v must have type τ
- 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.

			Rules Used
x	:	α_x	
y	:	α_y	
α_x	=	α_y	
β_0	=	Int	
α_y	=	Set[β_0]	
a	:	α_a	
z	:	α_z	
α_y	=	Set[β_2]	
α_z	=	Set[β_1]	
β_1	=	Set[β_2]	
α_a	=	Set[β_2]	

(b) (3 Points) What types will type inference obtain for the variables below?

x	:	Set[Int]
y	:	Set[Int]
z	:	Set[Set[Int]]
a	:	Set[Int]

(c) (4 Points) Consider the following complete program:

```
v := v + v;
w := @(w);
```

What types would type inference find for v and w? Explain.

Possible solution or hints: v: Only rule *(assign)* applies at the top level. Within the rule e uses (+), which is overloaded. Hence, *(adds)* and *(addi)* both apply. Via *(var)* we find that $\Delta(\mathtt{v}) = \textsc{Int}$ and $\Delta(\mathtt{v}) = \textsc{Set}[\tau]$ are both possible solutions. *(assign)* then also requires that $\Delta(\mathtt{v})$ is equal to the result of the $(+)$ expression, but that constraint is identical to the one we already have. Hence, v may be Int or Set$[\tau]$ for any type τ.

w: Only rule *(assign)* applies at the top level. Within that rule, e must come from rule *(choice)*. Combining *(choice)* with *(var)*, we find that we need $\Delta(\mathtt{w}) = \textsc{Set}[\tau]$. However, *(assign)* then also requires that $\Delta(\mathtt{w}) = \tau$. This means that we must require $\tau = \textsc{Set}[\tau]$, which fails the "occurs" check during unification. Our type system considers this term ill-typed.

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?

Possible solution or hints: The same as vertices in the abstract heap graph, i.e., program variables and heap-allocated memory locations.

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

Possible solution or hints: Andersen's algorithm uses two types of edges: points-to edges and inclusion edges. (For examples, see the slides.)

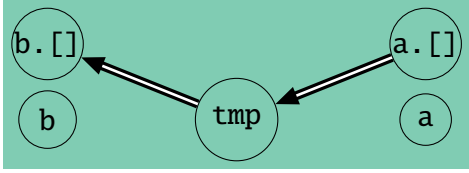
(c) (4 Points) Consider the following code in Teal:

```
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).

Possible solution or hints:

We need a temporary variable `tmp` as intermediary, since Andersen's closure rules only allow inclusion edges between dereferenced nodes and "regular" nodes



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
```

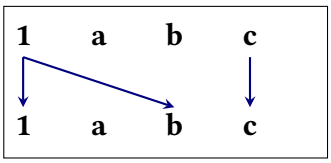
(a) (1 Point) Assume that we are performing this analysis to detect security flaws in an online banking system. Would you use a MUST or a MAY analysis? Justify your answer.

Possible solution or hints: Note that a MUST analysis would under-estimate the number of tainted nodes.

(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.

Possible solution or hints: MAY analysis:
First line: $1 \to 1, 1 \to x, y \to y, z \to z$
True branch: $1 \to 1, 1 \to y, z \to z$
False branch: $1 \to 1, x \to x, y \to y, y \to z$

In composition: $1 \to 1, 1 \to x, 1 \to y, y \to z, z \to z$

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`	$trans(E) = E[x \mapsto [0, 0]]$
`x := y + z`	$trans(E) = E[x \mapsto [y_l + z_l, y_r + z_r]$ where $[y_l, y_r] = E(y)$, $[z_l, z_r] = E(z)]$
`x := abs(y)`	$trans(E) = E[x \mapsto \begin{cases} [y_l, y_r] & \iff y_l \geq 0 \\ [-y_r, -y_l] & \iff y_r \leq 0 \\ [0, \max(-y_l, y_r)] & \iff y_l \leq 0 \text{ and } y_r \geq 0 \end{cases}$ where $[y_l, y_r] = E(y)]$

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?

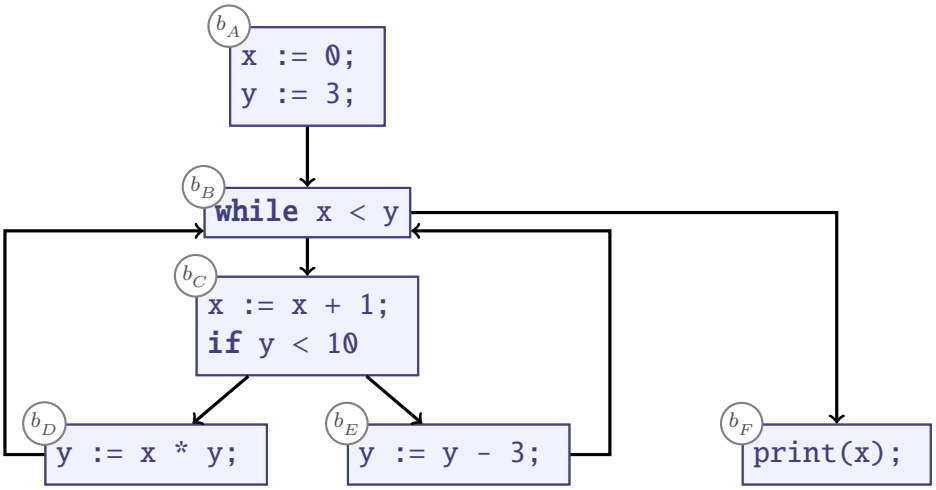
Possible solution or hints: No, cf. discussion of interval domain on the slides (infinite lattice height)

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)

Possible solution or hints: Using widening, e.g., by widening to all literals that appear in the program (guaranteed to be finite).

(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.

Possible solution or hints: Mainly different forms of precision:

Input dependence: static type analysis may not be able to determine e.g. that some variable only takes values of a subtype of its declared static type, since alternative code branches will never execute due to the program's configuration.

No false positives: all types come from observed values

May be faster for programs that are large but only short-running

(b) (3 Points) Give two arguments in favour of using a *static* analysis for the same task.

Possible solution or hints: Soundness, no run-time overhead, likely faster for most programs (depending on need for precision).

(c) (2 Points) Some dynamic analyses are affected by *perturbation*. Is perturbation a concern for such a dynamic type analysis? Explain.

Possible solution or hints: No perturbation. Perturbation takes place when the process of measuring interferes with the measurement. Dynamic type information does not depend on the process of measuring the type information.

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

$$\texttt{concat} : \textsc{array}[\tau] \times \textsc{array}[\tau] \to \textsc{array}[\tau]$$

that concatenates two arrays, i.e., `concat(a, b)` returns a new array that contains all the elements of `a` and `b`, in order.

The analysis is as follows:

- The **per-variable lattice** is $\mathbb{Z}^{\top}_{\bot}$, 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	$\mathit{trans}(E) = E[x \mapsto E(y)]$
x `:= new array[`τ`](`n`)`	$\mathit{trans}(E) = E[x \mapsto n]$
x `:= [`e_1`, ..., `e_n`]`	$\mathit{trans}(E) = E[x \mapsto n]$
x `:= concat(`y`, `z`)`	$\mathit{trans}(E) = E[x \mapsto E(y) +_{\bot} E(z)]$

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

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

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

Possible solution or hints: Yes: All transfer functions are monotonic (can be shown by examining them individually, and the lattice has height 3, i.e., is finite.

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

Possible solution or hints: No: *trans* does not satisfy $\mathit{trans}(E_1) \sqcup \mathit{trans}(E_2) = \mathit{trans}(E_1 \sqcup E_2)$ for concatenation: if the code concatenates a and b where $E_1 = [a \mapsto 1, b \mapsto 2]$ and $E_2 = [a \mapsto 2, b \mapsto 1]$, $E_1 \sqcup E_2$ will map a and b to $\top$. If we transfer first, we will get the result length 3 for both E_1 and E_2, i.e., a more precise result.

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`	: INT → INT	`thruster_on(`t`)` switches on thruster t and returns 1.
`thruster_off`	: INT → INT	`thruster_off(`t`)` switches off thruster t and returns 0.
`thruster_active`	: INT → 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 $0 \leq t \leq 3$.

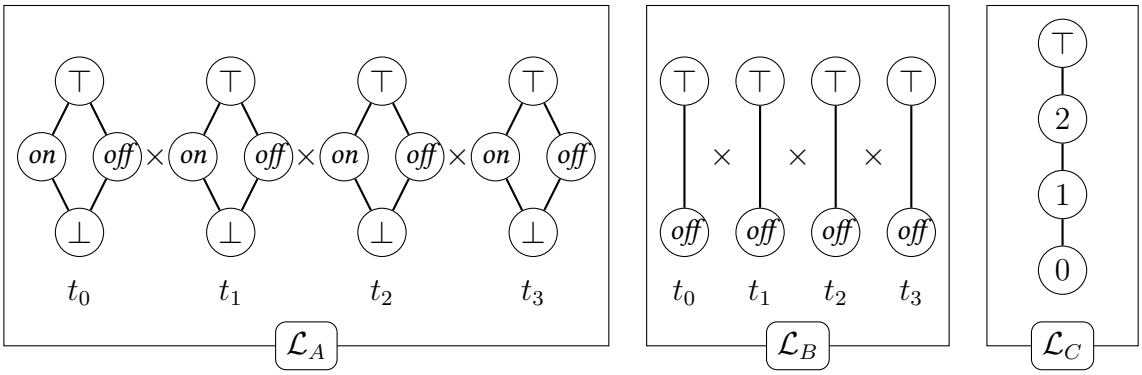
Your manager splits up the program analysis challenges into two parts:

- **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 are 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 $t_0 \ldots t_3$.
- $\mathcal{L}_A$ tracks whether the thruster is *on*, *off*, or in an unknown state ($\top$, $\bot$).
- $\mathcal{L}_B$ 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.

Possible solution or hints:

- $\mathcal{L}_A$: Most precise, can also support other analyses (cf. point (c))
- $\mathcal{L}_B$: More precise than $\mathcal{L}_C$, more efficient than $\mathcal{L}_A$, does not needlessly distinguish between "on" and "$\top$"
- $\mathcal{K}_C$: Fastest

(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.

Possible solution or hints: $\mathcal{L}_A, \mathcal{L}_B$: set suitable state in product lattice; $\mathcal{L}_C$: increase/-decrease state.

(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.

Possible solution or hints: **CT**: we may be able to support thruster operations with parameters that are not literals. **CI**: we may be able to yield a more precise result for `thruster_status` (for $\mathcal{L}_C$: only for state 0)

Question 10 (17 Points)

Answer the following questions:

(a) (1 Point) Points-To Analysis: Why might we choose Andersen's algorithm over Steensgaard's?

Possible solution or hints: Precision

(b) (1 Point) Why might we choose Steensgaard's algorithm over Andersen's?

Possible solution or hints: Speed, memory usage

(c) (2 Points) What is the purpose of a worklist in a worklist algorithm? Explain with a short example.

Possible solution or hints: Track tasks that the algorithm must still perform. This allows each individual task to generate an arbitrary number of new tasks. See slides for examples.

(d) (2 Points) In general, will the IFDS algorithm always attempt to compute the representation relation for the entire program under analysis? Explain.

Possible solution or hints: No, it only explores nodes in the exploded hypergraph that are reachable from the program entry point. The algorithm skips over dead code and does not examine the possible consequences of the property-of-interest at some program point unless it can show that the property-of-interest can hold at that program point.

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

Possible solution or hints: Tracks which call site can call which subroutine. When analysing a call site with dynamic dispatch, the call graph determines the set of possible callees for that call site. Increasing the quality of the graph can thus increase the quality of any interprocedural client analysis.

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

Possible solution or hints: Property of interest depends on values that are *assigned* to variables. Flow sensitivity allows modelling how the values are propagated by the program's control flow. (Example: interval analysis).

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

Possible solution or hints: Property of interest is not dependent on control flow, e.g., type inference in pure functional programming languages.

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

Possible solution or hints: Data flow analyses that are not distributive cannot be scaled up with IFDS. For these, we will merge different uses of the same subroutine if we do not exploit some form of context sensitivity, such as call-site sensitivity, to distinguish between how different call sites use the subroutine.

Example: Question 2(c): call-site sensitivity would allow us to assign different allocation sites to the nodes allocated via the calls in line 6 and thus determine that `b` and `c` cannot alias.