Static Analysis for Software Engineering (ECE 750-T05): Final (v1)

Instructor: Patrick Lam

April 16, 2021

This open-book final has 5 questions, each worth 20 points. Send me a PDF with your answers. You may consult any printed material (books, notes, etc), but you may not collaborate with anyone. You can ask clarification questions by email or on Piazza and I will answer them.

Question 1. Abstracts and Opinions

Part a (Your Favourite Paper), 10 points. Out of the papers we discussed this term, which did you appreciate the most? Describe what about it that you appreciated and why you think it is a worthy paper. Remember the criteria we used to evaluate papers: do we believe the result, and do we think that it is important.

Part b (Abstract Writing Practice), 10 points. For one of the papers from this term (could be the one above or not), write your own version of an abstract for that paper. Here's some tips on writing abstracts:

`https://www.acsmeetings.org/files/meetings/tips-for-writing-abstracts-annual-mtgs.pdf`

Question 2: Context-sensitivity

Consider an alias analysis which tracks whether pairs of local variables (ℓ_1, ℓ_2) may alias; you have to say that they may alias unless you know for a fact that they don't alias. Also consider the following program.

```
static void m1() {
  Object x = new Object();
  Object y = null;
  m3(x, y);
}

```

```
  static void m2() {
    Object x = new Object();
    Object y = x;
    m3(x, y);
  }

  static void m3(Object a, Object b) {
    Object c = new Object();
    if ( _ ) {
        c = b;
    }
    Object d = _ ? c : a;
    // here
  }
```

Write down, under worst-case assumptions, the aliasing information (a set of pairs) for `m3()` at point “here” if you know nothing about how it may be called.

Now, assume that the whole program consists of functions `m1`, `m2`, and `m3`. What is the aliasing information at point “here” using a context-insensitive analysis? And, what about with a context-sensitive analysis?

[If we had non-static methods we could start talking about object-sensitivity as well.]

Question 3: IFDS

The reference for this question is “Pipelining Bottom-up Data Flow Analysis” by Shi and Zhang (ICSE ’20). Your task is to produce an exploded super-graph for a null-dereference analysis on the code below as Coyote would.

```
int* foo() {
  int* p = malloc(sizeof(int));
  int* q = bar(p);
  int* r = p;
  return _ ? q : r;
}

int *bar(int *a) {
  int* b = null;
  ...
  int* c = _ ? a : b;
  int* d = _ ? bar(c) : c;
  return d;
}
```

What are the summaries of these functions? Which groups f_0, f_1, f_2 would they belong to?

Question 4: Superoptimization and Pruning

The reference for this question is "Dataflow-Based Pruning for Speeding up Superoptimization" by Mukherjee et al (OOPSLA '20).

Part a, 10 points. Consider function $f(x) = 2x + 3$ and specification $g(x) = x \mid H(x)$. Show how you can use input specialization and known bits to show inequivalence.

Part a, 10 points. Again consider function $f(x) = 2x + 3$ and now partially symbolic candidate $g(x) = x \oplus C$. Explain how the technique in Section 3.5 applies or does not apply to show inequivalence.

Question 5: Verifying Object Construction

The reference for this question is "Verifying Object Construction" by Kellogg et al, ICSE '20.

Consider the test case

```
BuilderTest.builder().x(0).y(0).build();
```

available at

`https://github.com/kelloggm/object-construction-checker/blob/master/object-construction-checker/tests/lombok/BuilderTest.java.`

Write down all of the type specifications that their tool infers to verify this code and explain how the tool typechecks this snippet.