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

Instructor: Patrick Lam

Target handin date: April 15, 2022

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 0. Class Participation (0 marks)

I have a fairly good idea of what class participation mark I'll assign to each of you. But, you tell me what grade you think you deserve, with justification, and I may adjust my grade accordingly. A+ means strong, sustained participation. A- means that you had a few decent contributions to the discussion every meeting but could've contributed just a bit more. B+ means that you said some things but might have benefitted from saying more. I'm not going to assign anything below a B+, and I wasn't really planning on giving As (i.e. I was going to give A+ and A- but not A); it is something in between A+ and A-, obviously.

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: Typing-related Bugs (20 points)

The reference for this question is "Well-Typed Programs Can Go Wrong: A Study of Typing-Related Bugs in JVM Compilers" by Chaliasos et al (OOPSLA 2021).

Carry out the bug classification process described on the paper on a bug in one of their listed compilers for a bug more recent than 9 April 2021. That is, do bug collection manually to find a bug, post-filter it to make sure it qualifies, and then do the analysis and categorization steps described in the paper. Your answer should contain a reference to the bug, include the symptoms and causes, and report on fixes and test cases.

Question 3: Test-Case Reduction (20 points)

The reference for this question is "Test-Case Reduction and Deduplication almost for Free with Transformation-Based Compiler Testing" by Donaldson et al (PLDI 2021).

Consider Figure 4 in this paper; it shows a series of transformations to a basic block, along with some inputs. Your task is to supply a similar program to the start program—make some changes—and apply 4 transformations of your choice to it. You should apply a different sequence of transformations than the sequence in the paper. Be sure to meet the preconditions of the changes that you apply. Indicate which (if any) blocks are dead as a result of the transformation.

Since we're not doing this with an actual compiler, it's hard to trigger a bug. But show me a hypothetical bug that your sequence of transformations would find, including the output with and without the transformations, and show me a minimized sequence that is 1-minimal. Show what it means for that sequence to be 1-minimal.

Question 4: SyRust

The reference for this question is "SyRust: Automatic Testing of Rust Libraries with Semantic-Aware Program Synthesis" by Takashima et al (PLDI 2021).

Part a (Experiments), 5 points. In RQ2 and RQ3, the authors turn off two of SyRust's features and evaluate how well SyRust does without these features. Name another SyRust feature that you could turn off for the purpose of an experiment. (It may be a strict subset or superset of one of the features in the paper, but obviously it should be somehow different). Describe your experiment and outline what you think you might find.

Part b (False negatives), 5 points. What's an example of a kind of program that SyRust cannot synthesize? Comment about how likely or unlikely this omission is to cause SyRust to miss errors in actual code.

Part c (Soundness), 10 points. SyRust sometimes synthesizes programs that have compilation errors and uses these errors to synthesize error-free refined programs. 1) Under which circumstances does SyRust synthesize programs with errors? 2) From what you can tell by reading the paper (you don't need to look at their implementation), should there be compilation errors in synthesized programs that SyRust doesn't handle? Justify your answer: if the answer is no, point at where in the paper supports your position, and if the answer is yes, provide a description of such a compilation error.

Question 5: Abductive Inference

The reference for this question is "Data-Driven Abductive Inference of Library Specifications" by Zhou et al (OOPSLA 2021).

Part a (Uninterpreted Functions), 5 points. Here, method predicates are "uninterpreted function symbols which have no intrinsic semantics"; and the paper includes two examples of formulas which are untethered to reality. Write down a formula which the SMT solver may return as a counterexample, but which doesn't make sense with the implied meanings of the predicates that we're using. You may use any of the predicates *hd*, *mem*, push, top, is_empty, and tail. To ensure that you're writing something different from what's in the text, you must use predicate tail.

Part b (Verification Interface), 5 points. Propose a verification interface $\mathsf{R}_{\mathsf{pop}}(\mathsf{s}, v)$ for a function that pops a value off stack s.

Part c (Positive Features), 10 points. An important part of the technique in this paper is creating positive training data. Create some client function (analogous to concat) which uses the pop function mentioned above, write down some inputs for it, the corresponding assignment to program variables, and a table analogous to Table 2 with the five feature vectors $\mathsf{R}_{\mathsf{top}}$, $\mathsf{R}_{\mathsf{tail}}$, $\mathsf{R}_{\mathsf{push}}$, $\mathsf{R}_{\mathsf{empty}}$, and $\mathsf{R}_{\mathsf{tail}}$. You get to choose which client function you write, but it should not be trivial.