Universitetet i Oslo
Institutt for Informatikk

Reliable Systems
Martin Steffen

IN [5/9]440: static analysis

Autumn 2018 **Handout** 28 Nov. 2018

Handout : Review check-up

Issued: 28 Nov. 2018

Review questions are meant as means to *check* yourself whether you understand (the background) of what is going on. The questions are largely informal, but you should think about whether you understand them and the answer. Try to formulate the answer yourself aloud, or write down notes what the answer is in your eyes, only that way you can check that you know it and not think that "Ah, I don't know how to say it, but it should be clear anyhow.". For instance, some questions below are rather general. As an example, for the section about procedures, there is the question "what problem are we dealing here with". Of course an answer is "we are dealing with procedures" but that's of course silly. What is meant here is:

1. what is the challenge when dealing with procedures (what is a procedure anyhow)
2. why is it harder than before?
3. what are possible solutions, etc.

So, the questions are meant so to make you *think* and *formulate* what's behind that section and only when you are able to formulate a (sensible) answer, it is a good sign.

Besides that: the review questions are not meant as a specification of the *pensum,* or a 1-to-1 repository of exam question. So just because a particular topic or question is not asked here or forgotten, it does not necessarily mean it cannot be asked or that it's not part of the curriculum (the curriculum is the material we covered in the lecture, including the presented talks).

1 Theory

Most of the things presented in the couse had a theoretical angle (motivated by practical needs, ...). Here I collect a few of general theoretical notions that came up.

1. What is an order? Give the definition? Where did orders came up in the lecture? What is a monotone function? What is a continuous function? What's the relationship between continuity and monotonicity?
2. That is a lattice? Give the definition? Where did lattices show up. What's the motivation for having lattices? Give an example or two of a *concrete* lattice?
3. What is a *fix-point*. What generalizations of fix-points we had? Give a definition. How many fixpoints "are there" (under what conditions...)
4. What *results* about fix-points did we have ("Fix-point theorems"). What is the importance of them? Where did they had a practical relevance?
5. What is *fixpoint iteration*. Who was it implemented?

2 Data flow

Some obvious review question I don't note down (like *what's a monotone framework* and what are the part of it. What is reaching definitions, etc. The central definitions of the material we covered there should also be thought about. Here I note more "background things" again. Also the discussion in the last lession shows that questions like " what is kill/generate" should be thought about.

1. Are data-flow analyses used in practice? What for?
2. What is "data" anyhow?
3. Explain how "equation solving" plays a part when doing data flow analysis? where do *fix points* enter the picture?
4. In the definition of monotone framework, they say that $\mathcal{F}$ is *closed* under composition. What do they mean?
5. An important ingredient in that chapter are lattices? Why are they crucial here?
6. If you are given the task of defining a *new* kind of data flow analysis/MF (or reconstruct one from the lecture whose details you have forgotten), how would you go for it, to determine whether its *forward* or *backward*? Or *may* or *must*?
7. When you (need to) define a specific data-flow analysis algorithms, what's the input and what's the output?
8. The while-language shown seems pretty impoverished. Can you think of arguments that the language is "good enough" for the purpose at hand? Can you think of other arguments that the language is too small. Certainly, the language is too small for practical purposes (that should be clear), but what about from the perspective of program analysis? Can you think why the language is not (or should not be) all of the story in this course?
9. unlike the other questions, this one is almost a technical one, about the labeling of the while language. In the slides it is mentioned that the labeling should be "unique" by which we mean, that one should not re-use a label, i.e., using each label only once. That's very natural and an obvious thing to do, as we intend the labels to *identify* different pieces of code inside the program (elementary blocks, as they are called). What happened with the analysis, if one marked two different blocks with the same label? Would an analysis be still "possible". Discuss.
10. What's the connection between data-flow analysis and fix-point theory?
11. Is "data-flow decidable"? Discuss what that question means and discuss under which circumstances it is decidable and/or not?

3 Control flow

Control-flow showed up in the introduction, but also in the (big) section with data-flow analysis and mononote frameworks,

- what is control flow analysis? We are doing this now in the chapter with types and effects, and have done similar analysis before. Maybe compare the *call-tracking analysis* we had at the beginning of the lecture to the CFA we have now. And of course: what is control flow anyway? How is the "control flow" captured in the presented "type-and-effect" system?

- Given a type-and-effect system, what is the *underlying* type system? What the relationship between the two?
- Why do we use the functional language to exercise that?
- In the type systems (and in general, if one thinks about it), I mentioned sometimes the notion of "meta-variables". What *is* a meta-variable? What is the difference to a "normal" variable? What's the connection with what is called in the book *type inference*
- What is a *type environment*?
- What is (in the context of writing a semantics as done in this Chapter) a *value*? Look at the grammar, to see what a value is, but what does it mean *intuitively*
- What is *subject reduction?* Why is it important, i.e., what is the *use* of subject reduction here/and in general.
- What's the difference between a small-step semantics and a big-step semantics (also known as natural semantics)? How is *non-termination* represented in each case?
- what is *sub-effecting?*
- Which role did the *constraints* play? What was the problem that motivated to make use of that concept?

4 Procedures

1. What are procedures? What is the problem that we are covering in that part?
2. What's the core problem of statically analysing procedures?
3. What (abstractly, the corresponding talk did not cover the details 2018) needs to be done for the semantics (and syntax) to take care of procedures?
4. What needs to be done wrt. the *control flow?*
5. Can one do inter-procedural analysis with the techniques we had before, i.e., without inventing something new?
6. What is the naive solution to the interprocedural control flow problem? What is problematic in that and why? How can one repair that?
7. What are *paths* (in general, resp. in our setting)? What are complete paths and what are valid paths?
8. What's inter-procedural flow?
9. How can one use *paths* to DFA for procedures? What are valid paths? How can they be formalized?
10. What's the problem with basing the data-flow analysis on analysing all paths through the CFG? What can be done to counter that problem?
11. That is the MVP approach in this context? What is a path? What is problematic in the MVP approach?

12. here's a tricky one: We have learnt about solving data flow equations (MFP) and (now) learnt about the MVP approach which specialiazes the MOP approach. Compare MFP and MOP (and compare MFP and MVP for procedures): Can they give different solutions, which solution is "better"?

13. What's an environment and a store? How do the concepts relate to the concept of "state" we already had for the simplistic while language?

14. What adaptions had been done for the transfer functions when it came to interprocedural flow graphs?

15. What's a "call-string"?

16. What's an *embellished* monotone framework? How can one *embellish* it?

17. What's a *context* here? Which role can it play?

1. What is data-flow analysis? For what kind of programs did we learn about data-flow?

2. Explain what role the *labels* in the programs we had played, and explain in that connection the words elementary blocks/control-flow graph.

3. What are the entry and exit points at an elementary block, and why we have to distringuish them, what's the transfer function in that context?

4. Now to shape analysis: what's the intention of shape analysis, i.e., what could it be used for? What's a "shape" in general? . For what kind of language did we define shape analysis?

5. What are the ingredients of a shape graph? (page 107)

6. Look at the figures of page 110 etc, described what you are seeing.

7. *the rest now starts to get tough:* Describe one of the figures describing the pre- and post-effect, for instance page 118, the effect of $x := y.sel$ (the toughest one would be 2.20).

5 Algorithms

1. What is chaotic iteration?

2. Explain the work-list algorithm? What is the work list? Does the algorithm terminate? If so: why?

3. explain intuitively the "time complexity" of the algorithm.

4. What is the theory behind the WL algorithm?

6 Type inference

1. What makes a "type system" an *algorithm?* Given a type system, how do you find out if it "is" an algorithm? If it's not, what are reasons why it's not, and if it's not: how can one potentially turn it into an algorithm?

2. What *is* type inference? Why actually is it important?

3. What is unification?[1] What role does unification play in this lecture? What is the decisive "trick"/insight for the type inference algorithm. What makes the type inference algorithm an "algorithm" as opposed to a "type system"?

4. What are type variables? What are meta-variables? What role do they play in this context?

5. what is a substitution? What is a unifier? what is the most general unifier? How are substitutions "ordered"? What is the importance of the *most general* unifier as opposed to just some unifier in this context.

6. Which of the rules of the type inference system is the most interesting, and why?

7. Sketch/explain the type inference rule for application.

8. in connection with type inference (but in the section of behavior analysis) there was the notion of type scheme? What is that? Why is it important (not necessarily (only) for behavior analysis, but in general?

9. which role do constraints play for type inference?

7 More things

1. what is the difference between an assignment $x := e$ in the while language and the $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ in the functional language?[2]

2. What is *substitution?* That one occurred in passing in the lecture, and I just assumed you know it, but make sure you do.

3. What is a *compositional* analysis (or what is "compositionality" in general)? Why is it an important concept? Is it connected with the software term "components"?

4. What is a type system? What is the purpose of those in general? Is a type system a static analysis? What are types anyway? What are *effects?* What are *annotated types* here.

5. What is a loop-invariant? What is partial correctness (and the opposite: *total* correctness), what is an assertion? Can you connect it to the notion of *fixpoint?* If you think about a type system (for instance one we have considered): is a typing judgment of the form

$$t : T \quad \text{or} \quad \Gamma \vdash t : T$$

to be interpreted in a *partial* or in a *total* correctness manner?

6. In the 3rd lesson and in the context of the while language, we presented the annotated "type" system (the formulation for the reaching definitions/assignments). What about the following alternative rule:

$$\frac{S : \mathsf{RD}_1 \to \mathsf{RD}_2}{\mathtt{while}[b]^l\,\mathtt{do}\,S : \mathsf{RD}_1 \to \mathsf{RD}_2}\ \textsc{While}'$$

Is this alternative rule "ok"?[3] If yes, which is "better", the one from the slides or this one here?[4] If it's not acceptable: can you repair it?

[1] Not covered in this lecture but: what is matching?

[2] In the abstract syntax *in the intro*, we omitted the let-construct, but in the (slightly) informal example illustrating the problems with CFA and higher-order functions we used it.

[3] What does it mean for a rule to be "ok", anyway?

[4] "Better" in a pragmatic sense.

Concerning the conditional: you may find it restrictive that the conditional rule insists on both branches to have the same type $\mathsf{RD}_1 \to \mathsf{RD}_2$. What about the following alternative:

$$\frac{S_1 : \mathsf{RD}_1 \to \mathsf{RD}_2 \qquad S_2 : \mathsf{RD}_1 \to \mathsf{RD}_3}{\texttt{if}[b]^l\ \texttt{then}\ S_1\ \texttt{else}\ S_2 : \mathsf{RD}_1 \to (\mathsf{RD}_2 \cup \mathsf{RD}_3)}\ \textsc{If}$$

Is the rule "ok". If it's ok: which rule is "better": this one or the one from the slides?. If it's not ok, can you repair it?