TDDD14/TDDD85 Some Exam Remarks

Victor Lagerkvist

This document contains some tips which will (hopefully) make the study period a bit smoother. It should not in any way be viewed as an indication for which problems might or might not appear during the exam.

Why are there no solutions to old exams?

Studying for an exam by learning from old solutions can easily become a form of rote learning where one ignores the bigger picture and only concentrates on memorising solutions. After graduation you will practially never have the opportunity to compare your own solution to a given solution. In fact, in general there are no perfect solutions, only approximations, and thinking that you cannot solve a problem without comparing it to a pre-given solution can lead to very bad habits of mind. For most types of exercises in this course there is not one, but typically many, good solutions, and you cannot in general verify whether your own solution is correct by comparing it to a pregiven solution.

Last, there is a *wealth* of exercises and detailed solutions in the course book(s), the lecture notes, and the tutorial compendium. And you can easily find more examples online if you so desire. Hence, if you think that it is easier to learn a certain technique (e.g. learning the minimisation algorithm for DFAs) by solving exercises with detailed solutions then you already have the possibility of doing so. Importantly, there is nothing magical about exam exercises which makes them stand out from other exercises which you have encountered in the course (a DFA is still a DFA, a regular expression is still a regular expression, and so on). Thus:

1. Focus on tutorials, exercises from the course books (both available online), and the lecture manuscripts, for learning a technique or a method.

2. Solve exam exercises to hone your skills and look through old exams to get a better sense of what is expected, knowledge-wise.

3. Ask me if you are unsure of what an acceptable exam answer would be in a certain scenario, e.g., what the write-up should consist of. However, keep in mind that the answer is typically very straightforward: your solution should include everything, including all step-by-step calculations that you have done, which allows us to verify your solution.

How can I verify my own solutions?

This is in general a difficult problem without a perfect solution. Thankfully it is substantially easier for many of the techniques and algorithms which we have developed during the course (otherwise, we as teachers would not be able to grade your solutions).

Example 1. *Assume that the goal is to construct a DFA for a given language. You struggle for a bit and then have a few ideas which you combine into a DFA. However, you are unsure of whether your solution is actually correct and would be an acceptable answer at an exam. In such a situation it is important to test your own solution, similarly to how you would write unit tests when solving a programming assignment. Thankfully, this is easy to do with a DFA. Since you were already given the language (in some description) you should be able to come up with a handful of example strings and simulate the behaviour of the DFA with those example strings. Here, just as in unit testing, it is important to include "edge cases", for example the empty string. Let us consider a few potential outcomes.*

1. *You cannot come up with any meaningful test data.*
2. *You come up with some example strings but is unable to simulate the behaviour of the DFA.*
3. *You simulate the DFA and find an example of a string which the DFA incorrectly accepts or rejects.*
4. *You simulate the DFA and do not find any strange behaviour.*

If you struggle with the first item then it is very likely that you do not understand the given language. Then you have no other choice than to go back to the drawing board. If you fail to simulate the DFA then either (1) the thing that you have drawn is not actually a DFA, or (2) you do not understand how a DFA operates. Hence, take a second look and compare your diagram with the definition of a DFA: what is the set of states? What is the transition function? Is it actually deterministic? Is the alphabet the same as for the given language? Recall that a DFA for a given state and a given symbol has exactly one choice.

Last, assume that you have found a potential counter example to your solution (e.g., a string which the DFA accepts but it should reject). In this case, after having verified a second time that the string is actually a counter example, try to use this string to fix your DFA. Do you have to add an accept state, remove an accept state, or add a new state together with some transitions? There are no general answers but it is in general much easier to fix a problem when you have a concrete counter example in mind. Also, if your DFA seems to be very far away from a working solution, then it might be better to go back to the drawing board, rather than trying to patch it.

The above hint works for essentially all exercises where you are asked to *produce* something, e.g., a minimal DFA, a regular expression, a context-free grammar, and so on. While it is typically preferable to verify something without the use of a computer program, there is an abundance of tools available online for e.g. simulating automata, which might also be helpful. However, I will not endorse any particular program or web site since I am unable to offer technical support for external programs. Some additional remarks:

1. If the task is to compute a regular expression for a given DFA then (1) generate a few example strings, and (2) compare the output of the DFA with your regular expression. If you cannot determine whether your regular expression matches a given string then you have to take a few steps back and resolve that problem first (e.g., by re-reading the definition of a regular expression and the language that it generates).

2. If the task is to minimise a DFA and you are unsure whether you have applied the minimisation algorithm correctly, then you should at least attempt to make sure that the resulting DFA generates the same language as before. Then check your calculations again: did you correctly mark all pairs of states in iteration 0 where one state is an accept state, and the other is not? Repeat this for iteration 1, iteration 2, and so on. Note that it is *much* easier to check your own solution if you have included sufficient details.

3. If the task is to construct a context-free grammar then generate a few example strings from your grammar using derivations and check whether they belong to the language in question.

Another class of exercises is where you are asked to *prove* something. This is, in general, harder to verify, but when we ask you to prove something we typically specify how it should be done. For example, proving that a language is not regular by using the pumping lemma or the Myhill-Nerode theorem.

Example 2. *Proving that a language is not regular can (often) be done with the inverted pumping lemma from lecture 6. Hence, assume that you have tried to use the inverted pumping lemma but are not sure whether the argument is correct. While there are no general guidelines that work in all circumstances (except the trivial advice of trying to go through each step and see if it a logical consequence of some previous steps) all proofs using the pumping lemma have a quite similar structure, even if the details of course may differ. Thus,* regardless *of the language in question your proof should contain statements along the lines of:*

1. *Let $p \geq 1$ be an arbitrary pumping length.*

2. *Let $s = \ldots$ be a string in the language of length at least p.*
3. *Let $s = xyz$ be an arbitrary partitioning of s into x, y, and z, where y is non-empty and $|xy| \leq p$.*

The dots here should of course be replaced by the string in question, which you can choose freely as long as (1) it is included in the language, and (2) it is of length at least p.

After having done this you have to proceed and show that you can always find an $i \geq 0$ such that xy^iz is not included in the language. If your proof does not *contain the above items then it is very likely incorrect.*

Naturally, giving a complete proof can still be difficult, but if you just make sure that your attempt contains the three items above then you can eliminate a lot of potential errors.

How do I know that my solution is sufficently detailed? What do I need to include?

This problem is related to the previous one but still a bit different in nature. We already have detailed solutions to similar exercises in the course book(s), the lecture notes, the tutorial compendium, and the homework assignments. If your level of detail is very different to those suggestions then it is very likely not sufficient. In particular, if you have a question along the lines of "Do I really need to include this part?" then the answer is very likely affirmative. We *never* deduct points for superflous details, but we *always* have to deduct points when important details are missing.

Similarly, if the exercise asks you to use a specific method (e.g, the GNFA-method) then you have to use the method in question, and based on existing solutions to similar problems it should already be clear which level of detail we expect.

What if I do not even understand the exercise?

Then you will have to go back to the original material and study the basic concepts (e.g., tutorial exercises, lecture notes, or the course book(s)). Try to find the most fundamental concept which you do not understand and concentrate on that. Yes, this can sometimes be more time consuming, but you will also pick up other skills in the process, increasing the likelihood of solving related (but not identical) exercises.