FIT2014 2022 exam feedback

Theory Of Computation (Monash University)

FIT2014 2022 S2 Final Exam Feedback

Overall, performance on the exam was better than last year.

Q1 (Propositional Logic) (2 marks)

Common mistakes:
- using a quantifier, treating it like predicate logic when it should be propositional logic
- answer not in CNF

Q2 (Predicate Logic) (1 mark)

Common mistakes:
- using strict inequality when it should be an inclusive inequality
- not starting with universal quantifier.
- some solutions along the lines of “Anyone who has classified at least as many stars as Annie Jump Cannon must be Annie Jump Cannon”
- incorrect use of the function stars()
- misuse of commas leading to ambiguity.

Q3 (Predicate Logic) (2 marks)

Generally done well.
Most common mistakes:
- some answers showed confusion about the difference between connectives (like implication, which relates propositions) and relations (like inequality/non-identity, which relate individuals)
- misplacing $\forall Y$
- wrong logic with the wrong inclusion of (X != Y), $(X \neq Y)$, or $\neg(X=Y)$.
- misuse of commas leading to ambiguity.

Q4 (Logic and Quantifiers) (2 marks)

Very well done. Performance was statistically consistent across the variants.

Q5 (Finite Automata) (4 marks)

Common mistakes included:
- producing NFA instead of FA (empty transitions, multiple transitions for the same letter from the same state)

- missing Start State and/or Final State
- wrong transitions for "backtracking" on reading **`b`**
- FAs incorrectly rejected/accepted certain substrings of the language (rejecting **`aaa`**, accepting **`aa`**), or all words with a certain property (rejecting words ending with **`aaa`** but accepting words ending with a different number of **`a`**s) or requiring strings to be significantly longer than the required minimum of 3.

Q6 (Regular expressions: proof by induction) (8 marks)

The question is clearly asking us to prove that for all n ≥ 1, **if** a string has exactly n occurrences of **`a`** and no occurrences of **`aa`**, **then** it matches the regular expression R_n. Too many students attempted to prove the opposite direction, i.e. **if** a string matches the regular expression R_n, **then** it has exactly n occurrences of **`a`** and no occurrences of **`aa`**. Both claims are true, but marks must come from answering the question.

Other mistakes:

- Some answers proved that the regular expression R_n itself only contains n **`a`**'s and no **`aa`**'s, which is trivial and not the point of this question.
- wrong base case starting number
- including a quantifier in the inductive hypothesis
- stating the inductive hypothesis but not using it later in the inductive step, instead giving general reasoning without using the inductive hypothesis.

Q7 (NFAs and regexps: an algorithm) (4 marks)

Common mistakes:

- not giving an algorithm (giving vague directions to merge NFAs without specifying an algorithm for how)
- using the regular expressions R and S directly
- using Kleene's theorem to create the regular expression for R ∪ S and converting it back into an NFA
- giving an algorithm that does not output a new NFA
- giving algorithm for different combinations of regexps (RS, R∩S)
- merging start states of A and B directly
- merging NFAs A and B together by combining all states
- giving an algorithm that used the wrong inputs and outputs, e.g., making a decider
- unnecessarily adding a single final state
- wrongly merged the final states of A and B directly.

Q8 (convert NFA to DFA) (6 marks)

Common mistakes:

- omitting the sink row
- omitting labels

- not creating a new row for each new state that appears on the right-hand side of the transition table
- not properly labelling the Start and Final states
- wrong start state (not considering the states reachable via empty transitions from the original start state)
- wrong states for some transitions
- still having some nondeterminism

Q9 (FA minimisation) (4 marks)

Well done overall.
Common mistakes:

- non-final states not combined correctly
- unlabelled start or final states
- some missed minimisation steps with the final states. Often only two out of the required three states were combined into a single final state.

Q10 (Give CFG for a language) (4 marks)

Common mistakes:

- trying to use a single non-terminal (S, X, E, etc…) to express the entire language. Many answers didn't seem to take account of the natural tripartite structure of the language or didn't effectively use it.
- Most answers seemed to show some understanding that each **`a`**/**`c`** should be paired with a corresponding **`b`**, but their rules often did not maintain correct structure without interleaving **`a`**s and **`b`**s (or **`b`**s and **`c`**s).
- using terminals on the left hand side instead of non-terminals.
- providing regular expressions instead of CFG for this question.
- Some CFG attempts only included rules to produce an equal number of abc's with only one extra **`b`** and could not produce strings like **`abbbbbc`** or **`aaabbbbbc`**.
- Some CFG attempts did not keep in mind the consecutive nature of **`a`**,**`b`**,**`c`** and could also produce **`bbbbcca`** or **`acbb`**.

Q11 (Proof of non-regularity using Pumping Lemma) (7 marks)

This is usually phrased as a proof by contradiction and it is crucial that a suitable word is chosen to arrive at said contradiction. The word needs to be generic in terms of a "pumping length" p or in terms of the number of states of a finite automaton recognising BIG MIDDLE.
Most common mistakes include:

- using just one or a few specific strings as examples
- using a generic word like **`a`**^n **`b`**^{2n} **`c`**^n, but not specifying where n comes from (which later on tends to make the case analysis invalid)
- using a word like **`a`**^n **`b`**^n **`c`**^n which isn't in the language to start with
- using a word like **`a`**^p **`b`**^q **`c`**^r, which is too generic to do the case analysis correctly

- making the word too short for the argument to work well as in **a**^{n/8} **b**^{3n/8} **c**^{n/8} where the pumping could occur anywhere (and this one is too short as well)

Some answers tried to use the wrong version of the pumping lemma, i.e. the one for context-free languages. That is doomed to fail, since we have shown in Q10 that the language is actually context-free.

Questions 12-13 concern the same language (BABSTAR).

Q12 (Draw parse tree) (2 marks)

Mostly done correctly.
A mistake sometimes made was to write a derivation instead of drawing a parse tree.

Q13 (CFG terminology) (1 mark)

Most answers correctly chose "regular"
Some answers chose "Chomsky Normal Form" which is incorrect.
Few answers chose "ambiguous" which is incorrect.
Many answers chose "leftmost" which is incorrect. It's not a term that can be applied to <u>grammars</u> themselves; it's a term that can be used for certain <u>derivations</u>.

Q14 (PDA: strings accepted & rejected) (2 marks)

Mostly well done.
Some mistakes:

- A few answers would try and define a language of accepted/rejected words instead of just simply giving two words as requested.
- wrongly including the empty string, while the question stipulated nonempty strings only.

A comment:

- Shorter and simpler examples tended to score better as it was easy to make a small mistake in a longer or more complicated string; I would encourage students to stick with the simplest examples they can think of for these style of questions.

Q15 (PDA: proof by induction) (7 marks)

This question was generally not well done. Many answers got some marks from some correct aspects of the structure of their inductive proof, but the inductive step itself proved difficult.
Most students had a good idea of the general structure of an induction proof, but there were some common errors here:

Base case:

- In the base case - not mentioning that both the stack is empty, and the PDA is in a final state. Both are part of the inductive hypothesis so both must be mentioned to prove it is true for a particular n.
- Proving the base case for n=1 rather than n=0. Some answers also claimed to be proving it for the case for n=0, but actually proved the n=1 case.

Inductive hypothesis:

- Not mentioning the stack is empty within their assumption
- Claiming “for all k” or not describing what k is when mentioned

Conclusion:

- Done quite well; however sometimes omitted.
- Occasionally omitted “for all n >= 0” or mention of the principle of induction

The key idea within the inductive step is to decompose the string a^(k+1) b^(k+1) into a suitable decomposition such that the inductive hypothesis can be used. Common mistakes here include

- Not providing a decomposition, but constructing a handwavey argument about equal numbers of **a**s and **b**s and the PDA structure. This is not a proof by induction.
- Rather than directly using the inductive hypothesis, analysing the structure of **a**^k **b**^k to argue about similar structures in **a**^(k+1) **b**^(k+1). From the inductive hypothesis we gain no information about *how* the string is accepted, only that it is. This is not a valid proof by induction.
- Attempting a decomposition, but one which does not lend itself to using the inductive hypothesis (e.g. **a**^k **a** **b**^k **b**).

Some students were able to construct a correct decomposition and attempt to reduce to the inductive hypothesis. Generally these answers scored quite highly.

- Most answers were structured well, linearly processing the decomposed string with reference to the PDA transitions, state, and stack
- Most students were able to appropriately argue about how the stack can be treated as if it were empty
- Often it was forgotten to mention that the PDA is in the start state; this is needed so we can treat the decomposed string like an initial input to the PDA.

Q16 (PDA: give CFG for it) (4 marks)

- Many students were able to describe some properties of the PDA using production rules; particularly, the requirement of **a**s on the left matched with **b**s on the right (captured often as S -> **a**S**b**) was well done.
- Most students noticed the empty string could be generated and added the appropriate production rule.
- However, more subtle properties of the PDA were often missed - specifically allowing more **a**s than **b**s, and allowing the concatenation of two accepted strings to be accepted. Thus, many grammars generated a subset of the required language.

- Some responses did not give a CFG or gave one with incorrect structure; for example, providing a regular expression, or providing a CFG with production rules with terminals on the left hand side, or providing a CFG with only non-terminals.

Q17 (Turing machine trace) (8 marks)

The Turing machine provided for this question had some unintended nondeterminism at State 4. In principle, the question was still valid, since nondeterministic Turing machines had been covered in lectures; in fact, the nondeterminism actually enabled a shorter computation path. Nonetheless, because the nondeterminism was not intended and possibly perplexing, any answer that had the trace correct up to the first point where nondeterminism arises was given full marks.

The question was mostly well done.
Common errors:

- Forgetting the state number
- In some cases, incorrect tape contents/tape head position

Q18 (Build a Turing machine) (6 marks)

Mistakes

- incorrect or missing handling for the case of the empty string. More generally, students should always remember to consider edge cases, and in particular, strive to design algorithms so that handling these cases falls out naturally from the general flow of the program/Turing machine, rather than needing special logic that complicates the algorithm (and in the context of the exam, exceeds the number of allowed states for the TM).
- multiple Accept states and/or making the Start state also an Accept state.
- A small number of answers gave non-deterministic TMs. This may be due to a lack of planning, making things up along the way, eventually leading to ambiguous transitions. Students should be aware that this is not a good strategy for designing algorithms.
- Producing a machine that looks more like NFA than a Turing machine
- Writing epsilon instead of a triangle to represent a blank tape cell.
- omitting information from an arrow.
- having outgoing arrows from the Accept state (which is treating the Accept State like a Final state in an FA).
- Using symbols other than a.
- confusion about even-length and odd-length input strings
- lack of necessary transitions to deal with blank tape cells
- using more than four states
- not specifying the Start State.

Q19 (Decidability/Undecidability) (1 mark)

Mostly well done.

Q20 (Decidability/Undecidability) (1 mark)

Mostly well done.

Q21 (Decidability/Undecidability) (1 mark)

Mostly well done.

Q22 (Decidability/Undecidability) (1 mark)

Mostly well done.

Q23 (Language classes, Venn diagram) (13 marks)

The cases that caused most trouble were:

- (h) — the "set of all Turing machines whose set of accepted strings is not recursively enumerable" — is actually equivalent to the empty language, since *any* Turing machine's set of accepted strings is recursively enumerable, *by definition*. So (h) was, in effect, a test on the definition of r.e.
- (l)
- (m)

(f) was less cut-and-dried than intended, as it depends too much on the encoding of the input. Any of P, CFL or Regular were marked as correct.

Q24 (Recursively Enumerable languages, Halting Problem, mapping reductions: proof) (8 marks)

Overall, this question was done poorly.
Mistakes included:

- confusion about the structure of the proof
- Ineffective use of course content that was only vaguely related to the question
- confusion about the difference between a decider and an enumerator
- trying to reproduce or adapt the proof from lectures that the Halting Problem is undecidable (but this was not what the question needed)
- referring to deciders for the Halting Problem (an impossibility)

- asserting without proof or justification that any language that had a computable mapping reduction to a recursively enumerable language is itself recursively enumerable. While this is true, the lack of any proof or justification fails to address the point of the question, so that it served more as a partial restatement of the question than a genuine answer.
- confusion about the direction of the mapping reduction, with some answers involving mapping reductions from the Halting Problem to another language (as opposed to the other way around).

Q25(a) (Boolean expression in CNF for graph problem) (4 marks)

Common mistakes:

- some answers are not in CNF
- some answers are incomplete
- some answers are not satisfiable (always False no matter what truth assignment is given)
- Some answers were too specific to the given graph instead of a general graph that could have any structure (i.e. focusing on brute forcing all combinations for this graph. While that technically works and gets full marks here (part (a)), it won't help with the next part which requires a general algorithm)
- constructing a wrong expression (some not in CNF) with clauses that correspond just to the examples given in the question.
- Constructed a wrong expression to incorrectly ensure, say, “at least one vertex is included”, “at most two vertices are included”, etc. (while the question is quite unrelated to such “at least / at most” conditions.)

Q25(b) (polynomial-time reduction to 2-SAT) (6 marks)

Mistakes included:

- Some answers were based on the overall shape of the graph instead of the condition of each edge. This may be manifested in trying to choose exactly two out of four vertices. Instead, the algorithm must be devised with no knowledge of the shape of the graph, but just purely based on the condition imposed upon both ends of each edge.
- This part is supposed to be an algorithm for any general graph. But some answers only gave an algorithm specifically aiming to solve the graph in part (a) only.
- not creating specific variables for the individual vertices. Without these, the problem is not truly translated into 2-SAT.
- looping over all vertices (instead of edges) when constructing the CNF.
- For an edge connecting v1 and v2, including only $(v1 \vee v2)$, omitting $(\neg v1 \vee \neg v2)$.

Q25(c) (complexity classification) (2 marks)

Common mistakes:

- using the wrong direction of reduction, reducing from 2-SAT to INDEPENDENT SET instead of from INDEPENDENT SET to 2-SAT.

- Some answers say that if a problem A is in NP, and there is a polynomial mapping reduction from A to a NP-Complete problem, then A is also NP-Complete. This means the direction of the reduction is wrong.
- just saying something like O(n^2) which is not, by itself, a time complexity class. That could mean O(n^2) to verify which would put the problem only in NP. It could refer to average or best case etc. An explicit class (i.e., the class P) should be given. Even just stating your algorithm runs in O(n^2) is insufficient without reasoning about 2-SAT being in P.
- not providing the complexity class and wrongly using the term 'Polynomial', which is ambiguous here: it may mean polynomial-time decidable (thus in P), or may mean polynomial-time verifiable (thus in NP).
- writing only about the running time of the algorithm in Q25(b) (for mapping the given graph to a CNF for 2-SATISFIABILITY) instead of the complexity classification for deciding INDEPENDENT VERTEX COVER.

Q26 (NP-completeness) (9 marks)

Mistakes included:

- not showing that EIS is in NP (which needs to be done by means of a verifier or proving that one exists).
- vagueness about the verification process (i.e., verifying the independent set property)
- trying to use the verifier to <u>solve</u> the problem (i.e., *finding* an independent set of vertices of the required size). This is not the role of the verifier.
- thinking that, in EIS, the "even" applies to the number of vertices in the input graph, whereas the evenness condition is supposed to apply just to the size of the independent set in that graph. The graph can have an odd number of vertices or an even number of vertices and that should not matter.
- misunderstanding the domain of the decision problem in the polynomial-time reduction, i.e., mapping G to G' rather than (G, k) to (G', k').
    - Typically this manifested as forgetting to map k or mapping k incorrectly.
- attempting to add n = |V(G)| isolated vertices to the graph. You must add k isolated vertices for this method to work. This issue was also tied into the above where students didn't map k correctly.
- devising mappings that worked in the 'forward' direction, i.e if G had an independent set of size >= k, but did not work in the 'negative' (G does not have an independent set of size >= k) or 'backwards' directions (G' has an independent set of size >= k'). Answers need to consider all required parts of the proof.
- not justifying the mapping reduction property in both directions (if and only if (⇔))
- not explicitly mentioning that the mapping reduction algorithm takes input (G,k)
- not specifying the input and output of the mapping reduction
- vagueness, or over-simplification, in justifying the mapping reduction property.
- not proving that the mapping reduction algorithm runs in polynomial time.
- bringing in the halting problem somehow (which is irrelevant to this question)