Formal Reasoning 2020
Solutions Test Block 2: Languages and Automata
(26/10/20)

Multiple choice questions

1. Which of the following four languages is different from the other three?

(a) is correct

(a) $\emptyset$
(b) $\emptyset^*$
(c) $\{\lambda\}$
(d) $\{\lambda\}^*$

Answer (a) is correct. The language $\emptyset$ is empty, which means that it contains no words. All other languages are equal to $\{\lambda\}$. If you concatenate zero words from the empty language, you get the empty word. If you concatenate zero or more times the empty word, it will still be the empty word.

1. Which of the following four languages is different from the other three?

(a) $(\{a, b\})^*$
(b) $(\{b, a, a\})^*$

(c) is correct

(c) $(\{a\}^* \cup \{b\}^*)$
(d) $(\{a\} \cup \{b\})^*$

Answer (c) is correct. The languages $(\{a, b\})^*$, $(\{b, a, a\})^*$, and $(\{a\} \cup \{b\})^*$ basically all state: choose either a or b and do that as often as you want. This gives all words over the alphabet $\{a, b\}$. The language $(\{a\}^* \cup \{b\}^*)$, however, states: take any amount of a's or any amount of b's. In particular this last language doesn't contain words like ab that contain a's and b's.

2. Is the intersection of two regular languages always regular?

(a) Yes, because if all words in both languages are regular, then the words in the intersection are also certainly regular.
(b) Yes, because the complement of a regular language is always regular, and $L \cap L' = \overline{\overline{L} \cup \overline{L'}}$.

(b) is correct

(c) No, because the intersection symbol '$\cap$' is not allowed in a regular expression.
(d) No, because a sublanguage of a regular language does not need to be regular.

Answer (b) is correct. De Morgan's law tells us that $L \cap L' = \overline{\overline{L} \cup \overline{L'}}$. But regular languages are closed under taking the complement. So if L and L' are regular, then $\overline{L}$ and $\overline{L'}$ are also regular. And regular languages are closed under taking the union, so $\overline{L} \cup \overline{L'}$ is also regular. Finally, we take another complement and we get that $\overline{\overline{L} \cup \overline{L'}}$ is indeed regular.

The first answer makes no sense because 'being regular' is not a property of words, but of languages.

The third and the fourth answer make no sense because the answer is 'yes'.

2. Is the complement of a regular language always regular?

(a) is correct

(a) Yes, because the regular languages are exactly the languages that can be recognized with a DFA, and it is easy to construct a DFA for the complement.

(b) Yes, because the complement of the complement is the language itself, so no information is lost by complementation.

(c) No, because the complement symbol '$\overline{L}$' is not allowed in a regular expression.

(d) No, because if the words in a language are regular, all words that are not regular will be in the complement.

Answer (a) is correct. It is true that the regular languages correspond to the languages generated by DFAs. By swapping all finite states and non-finite states in a DFA, we can compute the complement of a language with a new DFA, which by definition corresponds to a regular language again.

The second answer makes no sense because it assumes that if you apply an operation twice and you get back the original property, it should also be the case that the property holds after applying the operation only once, but there is no explanation why that should be the case. In general it doesn't hold for all operations.

The third and the fourth answer make no sense because the answer is 'yes'.

3. Consider the context-free grammar G_3:

$$S \to aS \mid Aa$$
$$A \to bS \mid Aa \mid \lambda$$

We want to show that $ba \notin \mathcal{L}(G_3)$ and for this we consider the following property as an invariant:

$$P(w) := w \text{ contains at least one } S \text{ or one } a$$

Does that work?

(c) is correct

(a) Yes, because after the first production step the words in a production will contain an a, which will not go away anymore in the rest of the production.

(b) Yes, because the property holds for S, and in each production step for S the symbol a is introduced.

(c) No, because the invariant is useless for this.

(d) No, because we have the rule $A \to \lambda$, and this rule does not introduce either an S or an a.

Answer (c) is correct. Note that property $P(w)$ is indeed an invariant.

- It clearly holds for S, because S contains at least one S.
- Now if we assume that v is a word such that $P(v)$ holds, this can be because v contains at least one S or at least one a.

- In the first case we again have to check some options: either there is a single S which is being replaced in the step, or there are more S's of which one is replaced, or a non-terminal A is replaced.
  - In the first case, v' has at least one a because both possible replacements of the S contain an a.
  - In the second case, only one S is replaced, so there will be at least one left in v'.
  - In the third case, the existing S will still be in v'. So in all cases we see that $P(v')$ also holds.
- In the second case, if v' is such that $v \to v'$, then $P(v')$ also holds, because a's are never removed.

However, $P(ba)$ also holds, so this invariant cannot be used to show that $ba \notin \mathcal{L}(G_3)$.

The first and the second answer make no sense because the answer is 'no'.

The fourth answer makes no sense because it is not related to the problem at hand at all.

3. Consider the context-free grammar G_3:

$$S \to aS \mid Aa$$
$$A \to bS \mid Aa \mid \lambda$$

We want to show that $ba \notin \mathcal{L}(G_3)$ and for this consider the following property as an invariant:

$P(w) :=$ if w contains the symbol b then it contains at least two a's

Does that work?

(a) Yes, because to get the symbol b in a word, you need to use the rule $A \to bS$, and any production that involves that rule needs to go through the rule $S \to Aa$ twice.

(b) Yes, because this property holds for all words in the language of G_3.

(c) No, because the invariant is useless for this.

(d) is correct

(d) No, because we have the rule $A \to bS$.

Answer (d) is correct. Note that property $P(w)$ is not an invariant. Let $v = A$ and $v' = bS$. Then clearly $P(v)$ holds because v contains no b's. And because of the rule $A \to bS$ it also holds that $v \to v'$. But $P(v')$ does not hold: it contains a b but no a's.

The first and the second answer make no sense because the answer is 'no'.

The third answer makes no sense because $P(ba)$ does not hold, which implies that if $P(w)$ was an invariant, it would have been a useful one.

4. Is there for all context-free grammars a right linear grammar that produces the same language?

(a) Yes, you can find such a grammar by converting to an automaton, and then converting back to a grammar.

(b) Yes, because all right linear grammars are context-free grammars.

(c) is correct

(c) No, because not all context-free languages are regular.

(d) No, because some rules can violate the condition of the context-free grammars being right linear.

Answer (c) is correct. We know that the language $\{a^n b^n \mid n \in \mathbb{N}\}$ is not regular. But it can be created with this context free grammar:

$$S \to aSb \mid \lambda$$

So this is a language for which we know for sure that a context-free grammar exists, but not a right linear one.

The first and the second answer make no sense because the answer is 'no'.

The fourth answer makes no sense because if a grammar is not right linear, the language still can have a right linear grammar as well. Take for example the language with grammar

$$S \to Sa \mid \lambda$$

The first rule is not right linear, but the language can also be defined with the right linear grammar:

$$S \to aS \mid \lambda$$

Therefore, the fact that some rules in a grammar are not right linear does not imply that the produced language does not have a right linear grammar as well.

4. Is there for all non-deterministic finite automata a deterministic finite automaton that recognizes the same language?

(a) is correct

(a) Yes, you can construct such an automaton using the power set construction.

(b) Yes, because you are not required to have non-determinism in a non-deterministic automaton.

(c) No, because in a non-deterministic automaton λ transitions are allowed.

(d) No, because non-deterministic automata are more efficient than deterministic automata.

Answer (a) is correct. This is basically Theorem 4.12 from the course notes, using the algorithm from Section 4.4.

The second answer makes no sense because this only explains that some NFAs are in fact already DFAs, but it doesn't explain why the other NFAs have an equivalent DFA.

The third and the fourth answer make no sense because the answer is 'yes'.

5. Are there languages for which every deterministic finite automaton has a sink (a non-final state for which each transition will loop back to that state)?

(a) Yes, when there is a word w for which any extension ww' will not be in the language.

(b) Yes, because each state needs to have a transition for each symbol.

(c) is correct

(c) No, because you can always split a sink in multiple states.

(d) No, because you can always use a non-deterministic automaton.

Answer (c) is correct. The idea of a sink is that you create a 'black hole' in a DFA: once you get in there, you can never get out anymore. This can be done using a single state with the 'sink property'. However, it can also be done using two non-final states q and q', where for each symbol s in the alphabet we have $\delta(q, s) = q'$ and $\delta(q', s) = q$. This has the same effect, but the 'looping' is not done directly to a single state, but indirectly between two states.

The first and the second answer make no sense because the answer is 'no'.

The fourth answer makes no sense because the question was about *deterministic* automata, and what is possible in a non-deterministic automaton is irrelevant for this.

5. Are there languages for which every non-deterministic finite automaton does *not* have a sink (a non-final state for which each transition will loop back to that state)?

(a) Yes, in the case that each word w has an extension ww' that is in the language, there cannot be a sink.

(b) Yes, in non-deterministic automata there never are sinks.

(c) No, one can always add a sink to an automaton without changing

(c) is correct

the language that is accepted.

(d) No, there are languages for which each non-deterministic finite automaton has a sink.

Answer (c) is correct. One can indeed always add a sink without changing the accepted language, because it will not lead to more accepted words.

The first and the second answer make no sense because the answer is 'no'.

The fourth answer makes no sense because you can always remove any sink without changing the accepted language.

Open questions

6. Give a regular expression for the language:

$$L_6 := \{w \in \{a, b\}^* \mid w \text{ does not contain } aab\}$$

Take for instance:

$$(b \cup ab)^* a^*$$

As long as we have blocks of only b's or ab's everything is fine. But these blocks leave out the fact that words can end on any number of a's.

6. Give a regular expression for the language:

$$L_6 := \{w \in \{a, b\}^* \mid w \text{ does not contain } abb\}$$

Take for instance:

$$b^*((a \cup ba)^* \cup (a \cup ab)^*)$$

Words may always start with any number of b's. After that they may stop. Or continue with blocks that have a's and b's, but in such a way that there are never two b's after each other, and the blocks end with an a. But we can also continue with blocks that have a's and b's, but in such a way that there are never two b's after each other, and the blocks end with a b.

This is equivalent to the simpler form:

$$b^*(a \cup ab)^*$$

7. Is the following language context-free?

$$L_7 := \{a^n b^n c^m \mid n, m \geq 0\}$$

If so, give a context-free grammar.

It is context-free because it can be generated with this grammar:

$$S \to Sc \mid A$$
$$A \to aAb \mid \lambda$$

7. Is the following language context-free?

$$L_7 := \{a^n b^m c^m \mid n, m \geq 0\}$$

If so, give a context-free grammar.

It is context-free because it can be generated with this grammar:

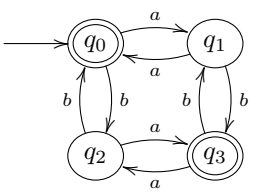
$$S \to aS \mid A$$
$$A \to bAc \mid \lambda$$

8. Give a deterministic finite automaton with the least number of states for the language:

$L_8 := \{w \in \{a, b\}^* \mid$ the number of a's in w is even iff the number of b's is even$\}$

(The abbreviation 'iff' stands for 'if and only if'.)

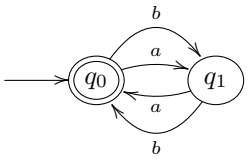
You may think that this is the proper solution:



This automaton generates the correct language. Words w for which it holds that the number of a's in w is even iff the number of b's in w is even, must have either an even number of a's and an even number of b's, or an odd number of a's and an odd number of b's. In the automaton the four states represent whether the number of a's and b's are even or odd.

- q_0 represents the situation that both numbers are even, which is allowed, so it is a final state.
- q_1 represents the situation that the number of a's is odd and the number of b's is even, which is not allowed, so it is not a final state.
- q_2 represents the situation that the number of a's is even and the number of b's is odd, which is not allowed, so it is not a final state.
- q_3 represents the situation that both numbers are odd, which is allowed, so it is a final state.

However, the automaton is not minimal. If you look at the words in this language you can see that the characterizing feature of this language is the fact that all words have an even length. And with that in mind it is easy to see that there is an automaton with fewer states that gives the same language:

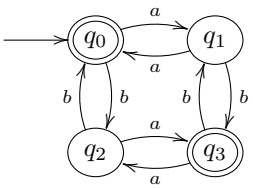


8. Give a deterministic finite automaton with the least number of states for the language:

$L_8 := \{w \in \{a, b\}^* \mid \text{the number of } a\text{'s in } w \text{ is odd iff the number of } b\text{'s is odd}\}$

(The abbreviation 'iff' stands for 'if and only if'.)

You may think that this is the proper solution:



This automaton generates the correct language. Words w for which it holds that the number of a's in w is odd iff the number of b's in w is odd, must have either an odd number of a's and an odd number of b's, or an even number of a's and an even number of b's. In the automaton the four states represent whether the number of a's and b's are even or odd.

- q_0 represents the situation that both numbers are even, which is allowed, so it is a final state.
- q_1 represents the situation that the number of a's is odd and the number of b's is even, which is not allowed, so it is not a final state.

- q_2 represents the situation that the number of a's is even and the number of b's is odd, which is not allowed, so it is not a final state.
- q_3 represents the situation that both numbers are odd, which is allowed, so it is a final state.

However, the automaton is not minimal. If you look at the words in this language you can see that the characterizing feature of this language is the fact that all words have an even length. And with that in mind it is easy to see that there is an automaton with fewer states that gives the same language: