Assignment 2

COMP30026 Models of Computation
School of Computing and Information Systems

Due: Friday October 11, 8pm

Aims

To improve your understanding of formal languages, automata, and grammars; to develop your skills in analysis and formal reasoning about complex concepts, and to practise writing down formal arguments with clarity.

Marking

Each question is worth 2 marks, for a total of 12. We aim to ensure that anyone with a basic comprehension of the subject matter receives a passing mark. Getting full marks is intended to be considerably more difficult; the harder questions provide an opportunity for students to distinguish themselves.

Your answers will be marked on correctness and clarity. Do not leave us guessing! We are trying to give you feedback, so you can improve. It is better to be clear and wrong; vague answers will attract few if any marks. This also means you must show your working in mechanical questions!

Finally, make sure your writing is legible! We cannot mark what we cannot read. (Keep in mind that the exam will be on paper, so this will be even more important later!)

Academic Integrity

In this assignment, ***individual work*** is called for. By submitting work for assessment you declare that:

1. You understand the University's policy on ***academic integrity***.
2. The work submitted is your original work.
3. You have not been unduly assisted by any other person or third party.
4. You have not unduly assisted anyone else.
5. You have not used any unauthorized materials, including ***but not limited to*** AI and translation software.

However, ***if you get stuck***, you can use the discussion board to ask any questions you have. If your question reveals anything about your approach or working, please make sure that it is set to "private".

You may ***only*** discuss the assignment in ***basic terms*** with your peers (e.g. clarifying what the question is asking, or recommending useful exercises). You may ***not*** directly help others in solving these problems, even by suggesting strategies.

Soliciting or accepting further help from non-staff is cheating and ***will*** lead to disciplinary action.

Submission – VERY IMPORTANT!

Submission is via Gradescope. Q4–Q6 are written questions and will be submitted as usual. But for **Q1–Q3**, you will submit your answers as Python programs. For submission, you will make use of the Python `automata` library, available at `https://pypi.org/project/automata-lib/`. To use the `visual` component (highly recommended), you must **first** install PyGraphviz by following the instructions at `https://pygraphviz.github.io/documentation/stable/install.html`.

We highly recommend installing the `visual` component, so you can visualize your automata before submission. If you do so, the library supports rendering automata automatically in **Jupyter** notebooks, which we **strongly** recommend. In any case, it will be easiest for you to **start on paper** and then transfer your work!

When you submit, the autograder will run some sanity checks to make sure your code is well-formed and passes very basic testing. More thorough testing will be conducted later.

Submitted code will only be assessed for correctness. Other attributes, such as readability or efficiency, will not be assessed. (Nonetheless, your code should finish almost instantaneously. If it is taking a noticeable amount of time, that is a sign that something is wrong.)

Q1 Regular Languages: Zero Blocks

Consider the language L of all binary strings with at least two blocks of 0's of even length, where we define a "block of 0's" to be a *maximal* nonempty substring of 0's. (By maximal, we mean that a block of 0's is not adjacent to any other 0's.) For example:

- The string 0000100 has two blocks of 0's of even length and is in L.
- The string 000 has only one block of 0's and is not in L.
- The string 00100100001000111 has three blocks of 0's of even length, so it is in L. The odd-length block of zeros is irrelevant.

For submission, upload a single Python source file `q1.py` which defines functions `q1a()` and `q1b()`.

- The function `q1a()` takes no arguments and must return an instance of `automata.fa.dfa.DFA`.

- The function `q1b()` takes no arguments and must return a string. The string is a regular expression following the syntax at `https://caleb531.github.io/automata/api/regular-expressions/`. Their syntax offers extra operators, but use only union (`|`), star (`*`), concatenation (juxtaposition), and grouping (`(` and `)`) in your submission. (The format will be checked will be automatically checked at submission time.)

Task A

Write a function `q1a()` which returns a DFA that recognises L.

Task B

Write a function `q1b()` which returns a regular expression for L.

Q2 DFA Construction: Injection

Given languages L_A and L_B over an alphabet Σ, define $\text{inject}(L_A, L_B)$ to be the language

$$\{xwy \mid xy \in L_A, w \in L_B\}.$$

That is, the language $\text{inject}(L_A, L_B)$ is the set of all strings which can be made by inserting, at any point, a string from L_B into a string from L_A.[1]

If L_A and L_B are regular, then $\text{inject}(L_A, L_B)$ is also regular. The goal of this question is to design a construction method that, given DFAs A for L_A and B for L_B, produces a NFA for $\text{inject}(L_A, L_B)$. You may assume that A and B have unique accept states.

For submission, upload a single Python source file `q2.py` which defines functions `q2a()` and `q2b(a, b)`.

- The function `q2a()` takes no arguments and must return a string.
- The function `q2b(a, b)` takes two arguments, each a DFA, and returns an NFA. The inputs will be instances of `automata.fa.dfa.DFA`, the return value must be an instance of `automata.fa.nfa.NFA`.

Task A

Your friend suggests the following method:

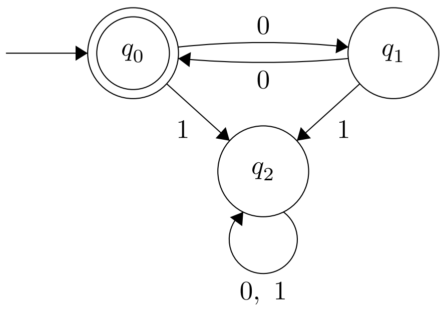
1. Take a copy of A and a copy of B
2. Use the same initial state and accept state as A
3. For every state q_a in A, add an ϵ transition to the start state of B and an ϵ transition from the accept state of B to q_a

The idea is that we process x in A and then jump to B and after processing w in B, we jump back to A and process y in A. For example, suppose L_A is the set of even-length[2] strings that are all 0s and L_B is the set of even-length

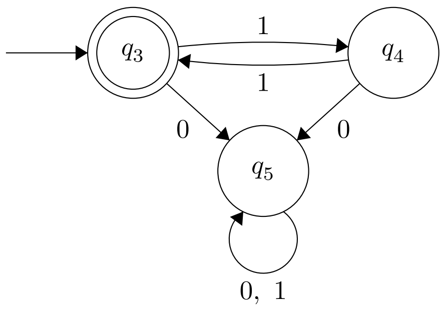
[1] More verbosely: $\text{inject}(L_A, L_B) = \{s \in \Sigma^* \mid \exists x, y \in \Sigma^* \, \exists w \in L_B (xy \in L_A \wedge s = xwy\}$.

[2] Recall that 0 is an even number.

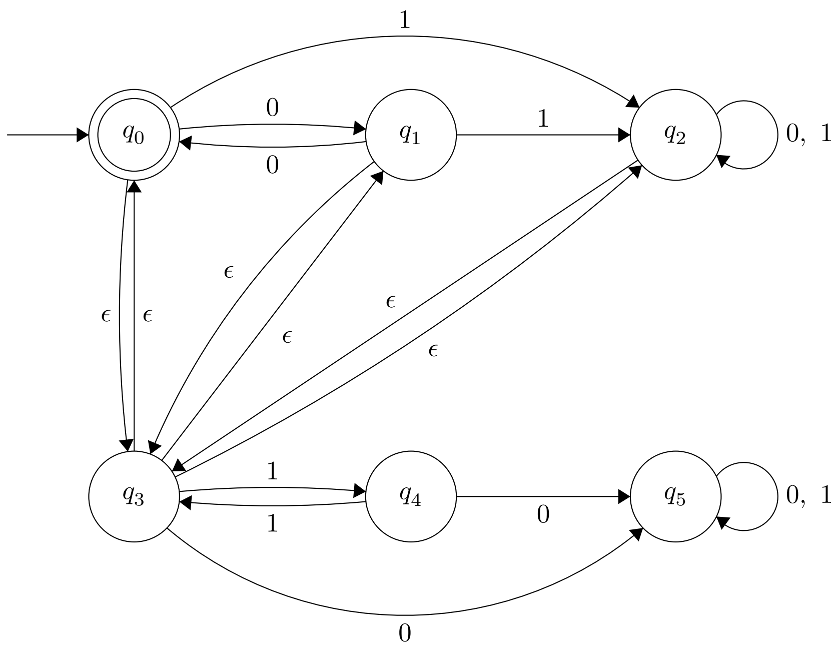
strings that are all 1s, i.e. $L_A = \{0^{2n} \mid n \geq 0\}$ and $L_B = \{1^{2n} \mid n \geq 0\}$. Then this is a DFA for A:



This is a DFA for B:



and this is the result of applying the above construction method:



Your task is to show that this construction method does not work for the above example. In particular, write a function `q2a()` which returns a string that is accepted by the NFA above but is not in $\text{inject}(L_A, L_B)$.

Task B

Write a function `q2b(a, b)` which returns an NFA for inject(L_A, L_B), where L_A is the language of the DFA `a` and L_B is the language of the DFA `b`.

Q3 Context-Free Languages: PDA and CFG Design

Consider the language

$$L = \{xy \mid x, y \in \Sigma^*, |x| = |y|, b \text{ occurs in y}\}$$

where $\Sigma = \{a, b\}$.

For submission, upload a single Python source file `q3.py` which defines functions `q3a()` and `q3b()`.

- The function `q3a()` takes no arguments and must return an instance of `automata.pda.npda.NPDA`.
- The function `q3b()` takes no arguments, and returns a context-free grammar as a list of pairs `(x, y)` such that `x` is a string consisting of a single letter (representing a variable), and where `y` is a list of strings (representing the right-hand sides of the rules). So, for example, the grammar

  $$\begin{aligned} S &\to aSb \mid A \\ A &\to \varepsilon \end{aligned}$$

  would become

  ```
  [
      ('S', ['aSb', 'A']),
      ('A', [''])
  ]
  ```

  Remember that, by convention, the start variable of a grammar is the left-hand side of the first rule!

Task A

Write a function `q3a()` which returns a PDA that recognises L.

Task B

Write a function `q3b()` which returns a context-free grammar for L.

Q4 Closure Properties

For all integers p, q and r, the language

$$\{a^n b^m \mid n, m \geq 0, np + mq = r\}$$

is context-free. Using this fact and the closure properties of context-free languages, prove that

$$L = \{a^n b^m \mid n, m \geq 0\}$$

is also context-free. In your proof, **do not** use any languages other than these or those derived from these using closure properties. (Proofs violating this requirement will receive 0 marks.)

Q5 Pumping Lemma for Regular Languages

Using the pumping lemma for regular languages, prove that

$$L = \{a^n b^m c^k \mid n, m, k \geq 0, nm = 2k\}$$

is not regular.

Q6 Pumping Lemma for Context-Free Languages

Let $\Sigma = \{a, b\}$ and

$$L = \{w \in \Sigma^* \mid \text{for all nonempty } s \in \Sigma^*, \text{ the string } sss \text{ does not occur in } w\}.$$

The language L is infinite. Using this fact and the pumping lemma for context-free languages, prove that L is not context-free.