After this tutorial you should be able to:

1. Express in English the language of a DFA/NFA.
2. Design DFAs/NFAs for languages specified in English.
3. Convert RE to NFAs
4. Form a DFA for the union or intersection of two DFAs using the product construction.
5. Devise algorithms for solving certain decision problems about DFAs/NFAs

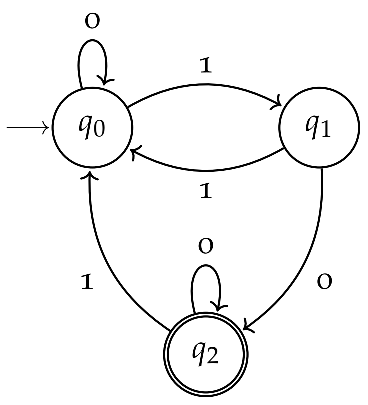
1 Warmup

Problem 1. Discussion. Look at the definitions of DFA and NFA.

1. What does "finite" refer to?
2. What does "automaton" refer to?
3. What does "deterministic" refer to?, i.e., in what sense are DFAs deterministic?
4. What does "non-deterministic" refer to? i.e., in what sense are NFAs non-deterministic?

2 DFAs

Problem 2. Consider the automaton drawn below:



1. Give $(Q, \Sigma, \delta, q_0, F)$ for this automaton.
2. What is language recognised by this automaton?

Problem 3. Draw the following automaton and give the language that it recognises:

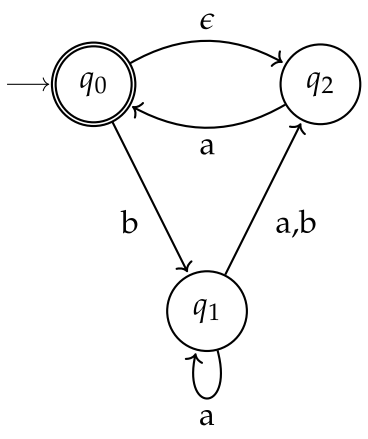
- $Q = \{q_1, q_2, q_3\}$ is the set of states.
- $\Sigma = \{b, c, d\}$ is the alphabet,
- q_1 is the start state.
- $F = \{q_2\}$ is the set of accept states,

and the $\delta : Q \times \Sigma \to Q$ is given by the following table:

	b	c	d
q_1	q_2	q_1	q_1
q_2	q_3	q_1	q_3
q_3	q_3	q_3	q_3

3 NFAs

Problem 4. Consider the following NFA:



Does the NFA accept:

1. ϵ
2. a
3. baba
4. baa
5. aaab
6. aabb

Problem 5. Draw an NFA for each of the following languages. The alphabet is $\Sigma = \{a, b, c\}$. You may use epsilon transitions.

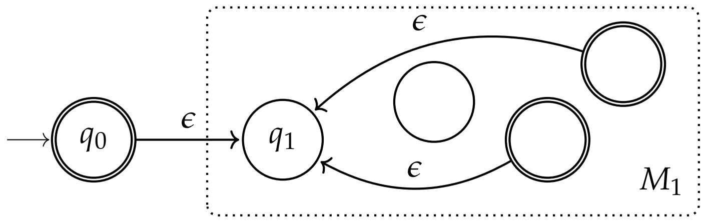
1. Strings that end with *aa* (use only 3 states)
2. Strings that contain the substring 'bbc' (use only 4 states)
3. Strings that start with a or end in c (use only 4 states)
4. Strings that start with a and end in c (use only 3 states)
5. Strings of length 4 starting with b
6. Strings of length at most 4.

4 From RE to NFA

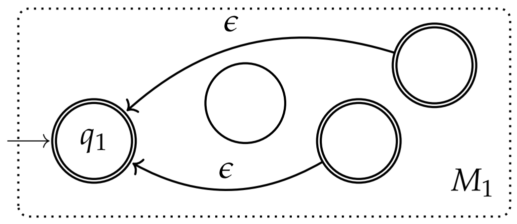
Problem 6. Use the construction demonstrated in lectures to construct an NFA for the following RE $0^\star 110^\star$

Problem 7. Use the construction demonstrated in lectures to construct an NFA for the following RE $((00)^\star(11) \mid 10)^\star$

Problem 8. Recall the construction of an NFA M recognising $L(M_1)^*$ given an NFA M_1:



Let us now consider the following construction instead, where we do not add a new start state, and instead make the original start state accepting:



Show that the second construction does not work, i.e., find a finite automaton M_1 for which the automaton M built using the second construction method does *not* recognise $L(M_1)^*$

Problem 9. Fix alphabet $\Sigma = \{a, b\}$.

1. Devise an NFA which recognises the language of strings containing two consecutive a's

2. Apply the union construction to build an NFA which recognises the language of strings with two consecutive a's or two consecutive b's
3. Write a DFA for this language (in the next tutorial you will use a systematic procedure for converting NFAs into DFAs!)
4. Write an RE for this language (in the next tutorial you will use a systematic procedure for converting NFAs into REs!)

5 Closure properties

Problem 10. In this problem you will show that the regular languages are closed under union. That is, if L_1, L_2 are regular, then so is $L_1 \cup L_2$.

The idea is to simulate both automata at the same time using pairs of states. This is called a *product* construction.

Here is the construction. We are given DFA $M_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ recognising L_i, $i = 1, 2$. We build a DFA $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = Q_1 \times Q_2$, i.e., $\{(s_1, s_2) : s_1 \in Q_1, s_2 \in Q_2\}$
- $q_0 = (q_1, q_2)$
- $F = \{(s_1, s_2) : s_1 \in F_1 \text{ or } s_2 \in F_2\}$
- δ maps state (s_1, s_2) on input $a \in \Sigma$ to $(\delta_1(s_1, a), \delta_2(s_2, a))$.

Then $L(M) = L(M_1) \cup L(M_2)$.

1. Argue why the construction is correct, i.e., why does M accept a string if and only if either M_1 accepts it or M_2 accepts it (or both).
2. Draw a DFA for the language of strings over alphabet $\{a, b\}$ consisting of an even number of a's or an odd number of b's.

Problem 11.

1. The product technique for building a DFA recognising the union of the languages of two DFAs can be slightly modified to get a DFA recognising the intersection of the languages of two DFAs. How?
2. Use the product technique to build a DFA for this language over $\{a, b\}$:

$$L = \{x \mid \text{there is no } b \text{ before an } a \text{ in } x \text{ and } |x| \text{ is even}\}$$

6 Algorithms

Problem 12. The *non-emptiness problem for DFAs* is the following decision problem: given a DFA M, return "Yes" if $L(M) \neq \emptyset$ and "No" otherwise.

Devise an algorithm for solving this problem. What is the worst-case time complexity of your algorithm?

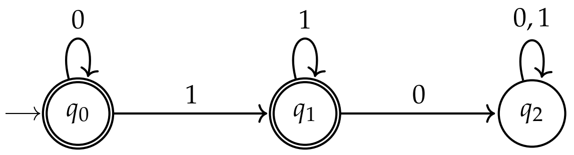
How would you adapt your algorithm to solve the non-emptiness problem for NFAs?

7 Additional problems

Problem 13.

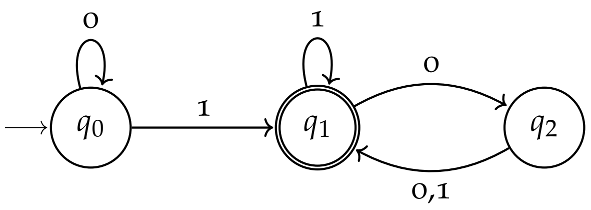
1. Construct a DFA that recognises the language of valid identifiers. A valid identifier begins with a letter or with an underscore '_' and contains any combination of letters, underscores, or digits, followed by a blank space.
2. How would you change your automaton if the length of the identifier (excluding the blank) should not be longer than 6 symbols?

Problem 14. Consider the automaton drawn below:



1. Is the automaton deterministic or non-deterministic?
2. What is the language recognised by this automaton.

Problem 15. Consider the automaton drawn below:



1. Give $(Q, \Sigma, \delta, q_0, F)$ for this automaton.
2. What is language recognised by this automaton?

Problem 16. Construct a DFA which recgonises the set of strings that are valid bracket expressions with at most 4 levels of nesting. For example, it should accept (((()))) or ()(()()) but not (() or)(.

Problem 17. Draw an automaton which recognises the language of strings over $\{b, c, d\}$ containing an even number of occurrences of the substring bcd. Recall that 0 is an even number.

Hint: think about what the automaton needs to remember as it moves through the input. Add a state for each of these conditions, then add appropriate transitions between them.

Problem 18. Prove that the following languages are regular, by drawing finite automata which recognise them. The alphabet for all the languages is $\{a, b, c\}$.

1. Start with b
2. End with b
3. Contain exactly one b
4. Contain at least one b
5. Contain at least one b and are of length exactly 2
6. Are of length at most 2
7. Are of length at most n for some integer n

Problem 19. Draw an automaton for comments in C or C++. These are strings which

1. begin with /*
2. end in */ (where the * cannot be the same as the first one)
3. have no other occurrences of */

Problem 20.

1. Give examples of strings described by the RE $(ab \mid ac)^\star$
2. Rewriting the RE in an equivalent form, construct two NFA (from the two regular expressions) which recognise the same language

Problem 21.

1. Write a regular expression that matches strings composed of: "-F␣" or "-f␣", followed by filenames consisting only any number of "a" (possibly none) followed by the extension ".tmp"

   Example: "-f aaaaa.tmp", "-f a.tmp", "-f .tmp" are all accepted, but "-fa.tmp", "-f b.temp", "-f aaa.pdf" are not.

2. Use the construction methods shown in lectures to convert the regular expression into an NFA.

Problem 22. Try construct a DFA M such that

$$L(M) = \{w | w \text{ contains an equal number of occurrences of the substrings 01 and 10}\}$$

For example, $101 \in L(M)$, but $1010 \notin L(M)$. If you cannot do so, what is the difficulty?

Problem 23. Discussion: Do you think every context-free language is regular?

Problem 24. A fundamental problem in string processing is to tell how close two strings are to each other. A simple measure of closeness is their Hamming distance, i.e., the number of position in which the strings differ, written $H(u, v)$. For instance, $H(aaba, abbb) = 2$ since the strings disagree in positions 2 and 4 (starting the count at 1). Positions at which the the strings differ are called *errors*.

Let $\Sigma = \{a, b\}$. Show that if $A \subseteq \Sigma^*$ is a regular language, then the set of strings of Hamming distance at most 2 from some string in A, i.e., the set of strings $u \in \Sigma^*$ for which there is some $v \in A$ such that $len(u) = len(v)$ and $H(u, v) \leq 2$, is also regular.

Hint. Take a DFA M for A, say with state set Q, and build an NFA with state set $Q \times \{0, 1, 2\}$. The second component tells you how many errors you have seen so far. Use nondeterminism to guess the string $u \in A$ that the input string is similar to, as well as where the errors are.

How would you change the construction to handle Hamming distance 3 instead of 2?

How would you change the construction to handle that case that we don't require $len(u) = len(v)$? e.g., $H(abba, aba) = 2$.

p.s. The problem of telling how similar two strings are comes up in a variety of settings, including computational biology (how close are two DNA sequences), spelling correction (what are candidate correct spellings of a given misspelled word), machine translation, information extraction, and speech recognition. Often one has other distance measures, e.g., instead of just substituting characters, one might also allow insertions and deletions, and have different "costs" associated with each operation. The standard way of computing similarity between two given strings is by Dynamic Programming, an algorithmic technique you will delve into in `COMP3027:Algorithm Design`.