EITQ : Homework

To be handed back on November 27, 9:00

1 Finite automata with ϵ-transitions

In this exercise, you will have to manipulate automata with ϵ-transitions. In nondeterministic automata as in the lecture, each transition triggers the reading of one letter of the word, and a transition can be used only when the corresponding letter must be read in the word. In automata with ϵ-transitions, some transitions are labelled with one letter, and some are with the empty-word ϵ. ϵ-transitions can always be used from the state from which they are outgoing, and do not trigger the reading of a letter of the word.

Definition 1 (Automaton with ϵ-transitions)**.** *An **automaton with ϵ-transitions** is a tuple $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ where:*

- *Q is a finite set, whose elements are called **states** of $\mathcal{A}$.*
- *Σ is an alphabet.*
- *I is a subset of Q, whose elements are called **initial states** of $\mathcal{A}$.*
- *F is a subset of Q, whose elements are called **final states** of $\mathcal{A}$.*
- *$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is a partial function, called the **transition function** of $\mathcal{A}$.*

In this definition, the only difference with nondeterministic automata is that the transitions can also be labelled with the empty-word ϵ. The following definition of ϵ-closure and extended transition function describes what "paths" are allowed when reading a word, and are used to defined the accepted language of an automaton.

Definition 2 (ϵ-closure of a set of states)**.** *Let $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ an automaton with ϵ-transitions, and $q \in Q$. The **ϵ-closure of** q, denoted as $E(q)$, is the set of states that are reachable from q by following ϵ-transitions in the automaton. In other words, $p \in E(q)$ if and only if there exist a sequence of states $q_1, ..., q_k$ such that:*

- *$q_1 = q$*
- *$\forall 1 \leq i \leq k$, $q_{i+1} \in \delta(q_i, \epsilon)$*
- *$q_k \in p$*

Definition 3 (Extended transition function of an automaton with ϵ-transitions)**.** *Let $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ an automaton with ϵ-transitions. The **extended transition function** of $\mathcal{A}$ is the function δ^* defined as follow:*

$$\forall q \in Q, \quad \delta^*(q, \epsilon) \stackrel{def}{=} E(q)$$

$$\forall q \in Q, \forall l \in \Sigma, \forall w \in \Sigma^* \quad \delta^*(q, lw) \stackrel{def}{=} \bigcup_{q' \in \delta(E(q), l)} \delta^*(q', w)$$

The language of an automaton with ϵ-transitions is the set of words that can reach a final state when read from an initial state.

Definition 4 (Language of an automaton with ϵ-transition)**.** *Let $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ an automaton with ϵ-transitions. The **language of** $\mathcal{A}$, noted $\mathcal{L}(\mathcal{A})$, is defined as follow:*

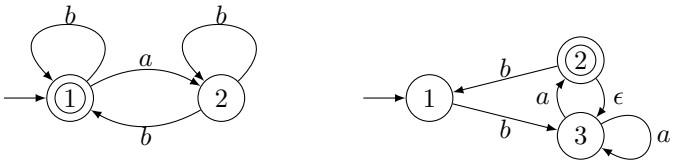
$$\mathcal{L}(\mathcal{A}) \stackrel{def}{=} \{w \in \Sigma^* \mid \delta^*(I, w) \cap F \neq \emptyset\}$$

where $\delta^*(I, w) \stackrel{def}{=} \bigcup_{q \in I} \delta^*(q, w)$.

In the following questions, unless explicitly mentioned, we consider nondeterministic finite automata with ϵ-transitions.

Question 1. For the following automata

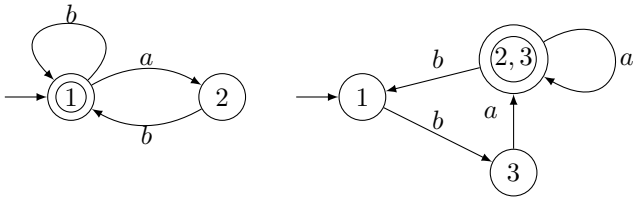
- Give the language they recognize.
- Say if they are deterministic or not.
- If they are not deterministic, give a deterministic automaton **without ϵ-transitions** that recognize the same language.



Answer: Call $\mathcal{A}_1$ the first automaton, and $\mathcal{A}_2$ the second one. None of these two automata are deterministic. $\mathcal{A}_1$ has two b transitions going from state 2, and $\mathcal{A}_2$ has an epsilon transition.

$\mathcal{A}_1$ recognizes the language of words on $\Sigma = \{a, b\}$ with no two consecutive a and not ending with an a. $\mathcal{A}_2$ recognizes $ba(a^*bba)^*$.

Here are determinized versions of these automata.



Question 2. For any language L, we write L^2 the languages of words that are the concatenation of two words of L, that is:

$$L^2 = \{uv \mid u \in L, v \in L\}$$

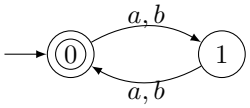
- Writing L_1 and L_2 for the languages recognized by the automata of Question 1, build an automata recognizing L_1^2 and an automata recognizing L_2^2.
- Describe a general procedure to build an automaton recognizing L^2 from an automaton recognizing L.

Answer: Draw two separate copies of the automaton recognizing L. The final state(s) of the first copy become non-final, and the initial state(s) of the second copy becomes non-initial. Add an ϵ-transition between all final state of the first copy to all initial state of the second copy.

Question 3. For the following languages, write an automata that recognize them.

- L_3 is the set of words over $\{a, b\}$ such that the parity of the number of a is the same as the parity of the number of b (so both even or both odd).
- L_4 is the set of words over $\{a, b, c\}$ such that the parity of the number of each letter is the same (so all even or all odd).

Answer: Here is a compact version of an automaton recognizing L_3. Another version with four states of $\{\text{even, odd}\} \times \{\text{even, odd}\}$ also works.



An automaton that can recognize L_4 is a one with 8 states : one for each possible triple of parities for a, b and c. When you read for example a b from a state even, odd, odd, you reach the state even, even, odd. This is what the following automaton does:

$$\Sigma = a, b, c$$

$$Q = \{\text{even, odd}\}^3$$

$$I = (\text{even, even, even})$$

$$F = \{(\text{even, even, even}), (\text{odd, odd, odd})\}$$

δ maps a pair q and a (resp. b and c) to the state q' where the first component (resp. second and third) is reversed.

2 Reduction from SAT to 3-SAT

We consider a finite set of Boolean variables $x_1, \dots x_n$. Recall that a SAT formula is obtained through the following grammar:

$$f_1, f_2 ::= x_i \mid \neg f_1 \mid f_1 \wedge f_2 \mid f_1 \vee f_2$$

A **literal** ℓ is a formula of the form x_i or $\neg x_i$, where $\neg$ stands for the negation. A **clause** is a formula of the form $\bigvee_i \ell_i$ where the ℓ_i are literals and $\vee$ stands for the Boolean "or". A formula in Conjunctive Normal Form (CNF) is a **conjunction** of clauses, of the form $\bigwedge_j c_j$ where the c_j are clauses and $\wedge$ stands for the Boolean "and". Recall that the problem SAT is: given an arbitrary formula, find if there is an assignment to true or false of the variables such that the formula evaluates to true. For example, the formula $x_1 \wedge \neg x_1$ is not satisfiable, while $x_1 \wedge \neg x_2$ is satisfiable by taking x_1 true and x_2 false.

A 3-SAT formula is a formula in CNF where each clause is of size ≤ 3. The aim of this exercise is to show that 3-SAT, the satisfiability problem on 3-SAT formulas, is **NP-complete**.

A boolean circuit is a graphical representation of a boolean function where we can copy variables 

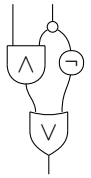
(as many times as required), perform their negation 

, their conjunction 

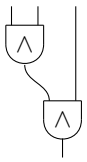
or their disjunction 

. The outputs of gates can be plugged in the inputs of other gates, but no feedback loop is allowed. A boolean circuit has as many input edges as the function has variables, and has one output edge (representing the function itself).

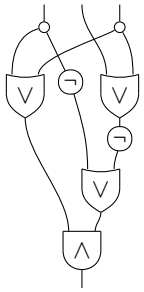
For instance, the following circuit represents the formula $(x_1 \wedge x_2) \vee \neg x_2$ (assuming variables are ordered x_1, x_2):



Question 1. Draw a boolean circuit for the formula $x_1 \wedge x_2 \wedge x_3$ with only binary gates (i.e. gates with exactly two inputs and one output).

Answer: We simply have to use $x_1 \wedge x_2 \wedge x_3 = (x_1 \wedge x_2) \wedge x_3$ for instance: 

Question 2. Draw a boolean circuit for the formula $\varphi := (x_1 \vee x_3) \wedge (\neg(x_2 \vee x_3) \vee \neg x_1)$.

Answer: 

Let's represent any output of the gates $\neg$, $\vee$ and $\wedge$ in a circuit as a fresh variable (i.e. a variable that was never used before).

Question 3. Suppose x_2 is the variable representing the output of the negation of x_1. Give a formula in CNF that describes the behaviour of x_2 w.r.t. x_1 (i.e. this formula should be true if and only if $x_2 = \neg x_1$).

Same question when x_3 is the output of the conjunction of x_1 and x_2; and when x_3 is the output of the disjunction of x_1 and x_2.

Answer: We give here a systematic approach, but it is not necessary for answering the question. A case-by-case analysis is sufficient.

One way to solve this problem is to first look at the truth table of the formula $\varphi_\neg$ we want to build:

x_1	x_2	$\varphi_\neg$
0	0	0
0	1	1
1	0	1
1	1	0

We could build $\varphi_\neg$ by doing a disjunction of the rows that give 1 as a result. However that would yield a disjunctive form. Since conjunction and disjunction are dual, we can instead build the disjunction of rows for $\neg\varphi_\neg$:

$$\neg\varphi_\neg = (\neg x_1 \wedge \neg x_2) \vee (x_1 \wedge x_2)$$

Negating the above and pushing the negations to the literals, we get the following CNF formula:

$$\varphi_\neg = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$$

Applying this approach to the other formulas, we get (up to some small simplifications):

$$\varphi_\wedge = (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3)$$
$$\varphi_\vee = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$$

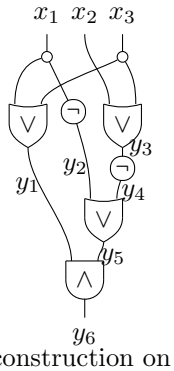
Question 4. Given a boolean circuit, explain how to build a formula in CNF that describes it (i.e this formula should be satisfiable if and only if the boolean circuit is). This new formula might have more variables than the original one.

Answer: Starting with a boolean circuit, we create a fresh variable for every output of a gate: we can annotate wires with these variables. We also annotate the inputs of the circuit with the corresponding initial variables. For each gate, we create a CNF formula that forces the value of its output depending on the value of its inputs (done in the previous question). To enforce the fact we are looking for a satisfying assignment, we also create a simple formula, consisting of the variable of the output of the whole circuit. The final formula φ is then the conjunction of all the gate-based formulas and that last variable.

For the last variable to be evaluated to 1 and φ to be satisfied, the penultimate variables have to satisfy the formula of the last gate, which means they'd also work as values for the inputs of the last gate. Propagating from the end to the start, φ is satisfiable iff the boolean circuit also is (a solution for the circuit forces all the values for the variables in φ, so it gives a solution to φ; a solution to φ gives in particular values for the initial variables that are a solution to the circuit).

Question 5. Apply this construction to the boolean circuit built in Question 2.

Answer: Let's first annotate the circuit:



The formula resulting from the above construction on this circuit is:

$$\begin{aligned}\varphi = y_6 \wedge \Big[(\neg y_1 \vee \neg y_5 \vee y_6) \wedge (y_1 \vee \neg y_6) \wedge (y_5 \vee \neg y_6)\Big] \wedge \Big[(y_2 \vee y_4 \vee \neg y_5) \wedge (\neg y_2 \vee y_5) \wedge (\neg y_4 \vee y_5)\Big] \\ \wedge \Big[(y_3 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)\Big] \wedge \Big[(x_1 \vee x_3 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \wedge (\neg x_3 \vee y_1)\Big] \\ \wedge \Big[(x_1 \vee y_2) \wedge (\neg x_1 \vee \neg y_2)\Big] \wedge \Big[(x_2 \vee x_3 \vee \neg y_3) \wedge (\neg x_2 \vee y_3) \wedge (\neg x_3 \vee y_3)\Big]\end{aligned}$$

Question 6. Using the answers to the previous questions, together with the assumption that SAT is **NP-complete**, show that 3-SAT is **NP-complete**.

Answer: We have defined a way to take any propositional formula ϕ and turn it into a 3-SAT formula φ, in 2 steps: first by building the circuit associated to the ϕ, then by giving a 3-SAT formula φ for this circuit. We already showed that φ is satisfiable iff the intermediary circuit is also satisfiable. It is direct to see that the circuit is satisfiable iff ϕ also is. We need to check that φ is of polynomial size w.r.t. the size of ϕ. Let's for instance count the connectors in both the formulas.

Notice that thanks in particular to Question 1, there is one gate in the circuit for each connector in the initial formula ϕ. As a consequence, the size of the circuit is polynomial (linear actually) in the size of ϕ. Each gate defines a formula that uses at most 10 connectors (worst case is the conjunction). All these have to be put in conjunction at the end, so that's one more connector per gate. φ hence has at most 11 times as many connectors as ϕ does, so its size is indeed polynomial in the size of ϕ.

We have shown here that there is a polynomial reduction from SAT to 3-SAT. Since SAT is **NP-complete**, 3-SAT is **NP-hard**. It is easy to see that 3-SAT is also in **NP** (it is even a subset of SAT). As a result 3-SAT is **NP-complete**.