EITQ TD: Advanced Complexity Theory

Renaud Vilmart
renaud.vilmart@inria.fr

October 17, 2024

1 SAT is NP-Complete

Question 1. Write a (non-deterministic) automaton that recognises a word $w = (w_1, ..., w_n)$ on the alphabet $\Sigma = \{0, 1\}$ iff the word ends with a 1.

We consider a finite set of Boolean variables $x_1, \dots x_n$. 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 **conjunction** of clauses is a formula of the form $\bigwedge_j c_j$ where the c_j are clauses and $\wedge$ stands for the Boolean "and". The problem SAT is as follows: given a formula find if there is an assignment to true or false of the variables such that the formula evaluates to true.

We want first to build a SAT formula that encodes the automaton and the input word w (and we want that process to work for any automaton and word w). We assume the automata we consider always terminate after $p(n)$ steps where n is the size of w.

We define variables $t_{i,k}$ ($1 \leq i \leq n+1$, $1 \leq k \leq p(n)$) that are true iff at step k, the next letter to read is w_i. $t_{n+1,k}$ means we are done reading the word.

Question 2. Write a formula φ_0 ensuring that exactly one of $\{0, 1, \epsilon\}$ is read at each step.

$\varphi_0 := t_{1,1} \wedge \bigwedge_k \left((\bigvee_i t_{i,k}) \bigwedge_{i \neq i'} (\neg t_{i,k} \vee \neg t_{i',k}) \right)$. The first clause ensures we start at the first letter, the second that at least 1 symbol is read, and the last one that no two different letters are read at the same time. φ_0 uses $1 + p(n)n(n+1) = \mathcal{O}(n^2 p(n))$ literals.

We define variables $Q_{q,k}$ that are true iff we are at state q at step k.

Question 3. Write a formula φ_1 ensuring that we are at a single state at each step.

$\varphi_1 := \bigwedge_k \left((\bigvee_q Q_{q,k}) \bigwedge_{q \neq q'} (\neg Q_{q,k} \vee \neg Q_{q',k}) \right)$. φ_1 uses $p(n)|Q|^2$ literals.

Question 4. Write a formula φ_2 stating which are the initial states.

$\varphi_2 := \bigvee_{q_i \in I} Q_{q_i,0}$. φ_2 uses $|I| \leq |Q|$ literals.

Question 5. Write a formula φ_3 stating that we must finish on an accepting state.

$\varphi_3 := \bigvee_{\substack{q_f \in F \\ k}} t_{n+1,k} \wedge Q_{q_f,k}$. φ_3 uses $2|F|p(n) \leq 2|Q|p(n)$ literals.

Question 6. Write a formula φ_4 encoding the transitions of the automaton.

$\varphi_4 := \bigwedge_{q,i,k} \left(Q_{q,k} \wedge t_{i,k} \implies \left(\bigvee_{(q,w_i),q' \in \delta} (Q_{q',k+1} \wedge t_{i+1,k+1}) \bigvee_{(q,\epsilon),q' \in \delta} (Q_{q',k+1} \wedge t_{i,k+1}) \right) \right)$. This uses less than $|Q|np(n)(2 + 2|Q|)$ literals.

Question 7. Justify why we can assume $|Q| = \mathcal{O}(1)$ (where $|Q|$ is the number of states of the automaton). What is then the size of the formula $\varphi := \varphi_0 \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$ w.r.t. n?

Justify why φ encodes the automaton into a SAT problem.

We consider the size of the program as fixed, while we make the size of the input vary, so $|Q| = \mathcal{O}(1)$. Then, the size of φ is $\mathcal{O}(n^2p(n)) + \mathcal{O}(p(n)) + \mathcal{O}(1) + \mathcal{O}(p(n)) + \mathcal{O}(np(n)) = \mathcal{O}(n^2p(n))$, so the size of φ is polynomial in n.

Question 8. What would we need to modify to make this work for a turing machine? Assuming it can be done, with a formula of size polynomial in n the size required on the tape to encode the input, what does that tell us about the complexity of the problem SAT?

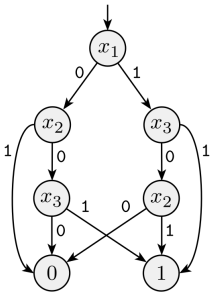
We would need to encode the position of the head at any given time, and the symbols written on the tape, at any single time. We would need to encode the initial position of the head, the intial content of the tape, the fact that at any step there is exactly one symbol per cell of the tape, the head is at exactly one position, and that the content of the tape can only change where the head is. The transition would have to account for the position of the head, the current state in the machine, and the content of the tape.

By definition of **NP** problems as problems that can be solved by a polynomial-time non-deterministic Turing machine, any **NP** problem can me reduced to a SAT problem. Assuming the size of the SAT formula remains polynomial, the reduction is polynomial. This means SAT is **NP-hard**. SAT is also in **NP**: checking if a valuation of the variables satisfies the formula can be done in polynomial time. SAT is hence an **NP-complete** problem.

2 Branching Programs

A branching program (BP) is a directed acyclic graph, where nodes are labelled by variables from V, except for two output nodes, labelled respectively by 0 and 1. The root node has no incoming edge. Each non-output node has two outgoing edges, labelled respectively 0 and 1. They represent boolean functions. For instance, the BP on the right represents the boolean function represented by the formula:

$$(\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge \neg x_3 \wedge x_2) \vee (x_1 \wedge x_3).$$



Question 1. Build a BP representing the following formulas:

- x_1
- $\neg x_1$
- $x_1 \wedge x_2$
- $x_1 \vee x_2$

Question 2. Suppose we have two BPs representing resp. formulas φ_0 and φ_1. Give a BP representing each of the following formulas:

- $\neg\varphi_0$
- $\varphi_0 \wedge \varphi_1$
- $\varphi_0 \vee \varphi_1$

Apply this construction on the formula above. Do we end up with the same branching program?

We denote by eqBP the problem of deciding the equivalence of two given BPs (i.e. if they represent the same boolean function).

Question 3. Considering that UNSAT is **coNP-complete**, show that eqBP is **coNP-complete**. Do you think eqBP is in **BPP**?

Provided a formula representing a boolean function, we can create a BP that represents the same function. The BP has size polynomial (linear even) in the size of the formula. The formula is unsatisfiable iff the BP is equivalent to a BP representing the null boolean function (such a BP is trivial to build). Hence, UNSAT

reduces to eqBP, hence since UNSAT is **coNP-complete**, eqBP is **coNP-hard**. eqBP is also in **coNP**, since checking whether two BPs differ on a given valuation is easy. Hence eqBP is **coNP-complete**.

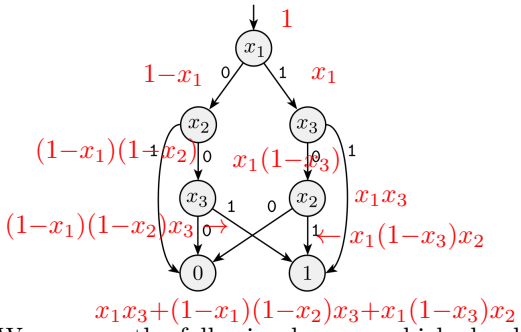
A read-once branching program (ROBP) is a branching program with the additional constraint: on each path from the root to an output node, each variable is queried at most once. We want to prove here that eqROBP (deciding the equivalence of two ROBP) is in **BPP**.

Question 4. Suppose we have two *real* variables y_1 and y_2. Find a real-valued polynomial P such that $P(y_1, y_2)$ gives the same value as $y_1 \wedge y_2$ when restricting to boolean values. Same question with $\neg y_1$ and $y_1 \vee y_2$.

$y_1 \wedge y_2 \rightsquigarrow y_1 \times y_2, \quad \neg y_1 \rightsquigarrow 1 - y_1, \quad y_1 \vee y_2 \rightsquigarrow y_1 + y_2 - y_1 \times y_2$

Question 5. Given an ROBP, how do we build a real-valued polynomial P that behaves like the boolean function that it represents when restricting to boolean values? What polynomial do you obtain from the BP above? What is the degree of P in general (for each variable), and is the size of P polynomial in the number of variables?

Annotate root with 1. Given a node with variable x annotated with polynomial P, annotated 0-edge with $(1-x) \times P$ and 1-edge with $x \times P$. Given node where all incoming edges are annotated by polynomials P_1, ..., P_n, annotate the node with $P_1 + ... + P_n$. Keep the polynomial obtained at the 1-leaf. The degree of the resulting polynomial for each variable is 1. The polynomial can be of exponential size.



We assume the following lemma, which should be somewhat intuitive:

Lemma 1. *Let $\mathcal{F}$ be a finite field with f elements, and $P \in \mathcal{F}[X_1, ..., X_m]$ non-zero, where each variable has degree $\leq d$. If $a_1, ..., a_m$ are selected at random in $\mathcal{F}$, then:*

$$\Pr[p(a_1, ..., a_m) = 0] \leq \frac{md}{f}$$

Question 6. Using this lemma, prove that eqROBP is in **BPP**.

- Pick $a_1, ..., a_m$ at random from some field with at least $3m$ elements
- Compute $P(D_1)(a_1, ..., a_m)$ and $P(D_2)(a_1, ..., a_m)$ without computing the actual polynomial
  - If $P(D_1)(a_1, ..., a_m) = P(D_2)(a_1, ..., a_m)$ accept
  - If $P(D_1)(a_1, ..., a_m) \neq P(D_2)(a_1, ..., a_m)$ reject

If $D_1 \equiv D_2$, then we accept with probability 1. Otherwise:

$$\Pr[P(D_1)(a_1, ..., a_m) = P(D_2)(a_1, ..., a_m)] = \Pr[(P(D_1) - P(D_2)(a_1, ..., a_m) = 0] \leq \frac{1}{3}$$

Further Information on Branching Programs

It is not known whether eqROBP is in **P** or not.

A binary decision diagram (BDD) is a ROBP where variables always appear in the same order. Checking the equivalence of two BDDs with the same variable ordering is in **P**.

3 3-COLOUR is NP-complete

Here we assume that 3-SAT is **NP-complete** (proven in the next exercise with the additional result that SAT $\preccurlyeq_P$ 3-SAT). Given a graph $G = (V, E)$, a k-colouring of the graph is an assignment $C : V \to [\![0, k-1]\!]$ such that $(u, v) \in E \implies C(u) \neq C(v)$, i.e. there are no adjacent vertices with the same colour. We mean to show in this exercise that 3-COLOUR (deciding if a graph admits a 3-colouring) is **NP-complete**.

Question 1. Show that 3-COLOUR $\in$ **NP**.

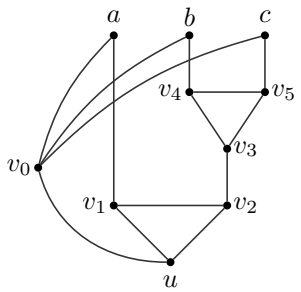
Provided the assignment C, checking that it is a 3-colouring can be done in linear time in $|E|$ the number of edges.

We then move on to show that 3-SAT $\preccurlyeq_P$ 3-COLOUR. Hence, consider a SAT formula φ_0. We want to build a graph from φ_0 that is 3-colourable iff φ_0 is satisfiable.

Question 2. Start building a graph with three nodes T, F and B all connected together. For each variable p_i in φ_0, create two vertices labelled p_i and $\neg p_i$. How do you connect them to the rest of the graph to make sure that one colour corresponds to evaluating to True, and another colour to evaluating to False, consistently?

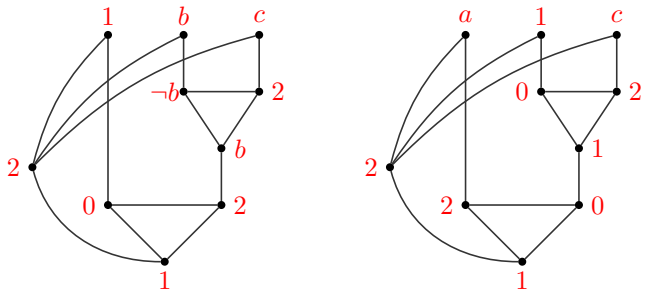
The colour $C(T)$ corresponds to the "True" evaluation, and $C(F)$ to the "False" evaluation. For each variable p_i, vertices p_i, $\neg p_i$ and B should be pairwise linked.

Question 3. Consider the following piece of graph:

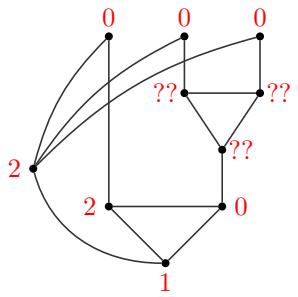


Show that the only valid 3-colourings are the ones where $\exists x_i \in \{a, b, c\},\ C(x_i) = C(u)$.

W.l.o.g. suppose that $C(u) = 1$ and $C(v_0) = 2$, so that the colours of a, b and c are in $\{0, 1\}$. When 1 of $\{a, b, c\}$ is 1, we can exhibit a valid 3-colouring:



When trying to find a 3-colouring when all top vertices are given color 0, we eventually get stuck:



Question 4. Using the above "gadget" for every clause in φ_0, complete the graph so that it has a 3-colouring iff φ_0 is satisfiable.

Node u will be common for every clause: that will be vertex T, and v_0 will already be there as B. Every clause will create 5 vertices v_1, ..., v_5 as in the gadget (with u being T), nodes a, b and c will simply be the vertices associated to the corresponding literal in the clause.

Question 5. Show that 2-COLOUR (the problem of deciding if a given graph admits a 2-colouring) is in **P**.

2-colourable graphs are exactly bipartite graphs, a property that is easy to check.