Complexité avancée - TD 5

Benjamin Bordais

October 21, 2020

Exercise 1 Family of circuits

Definition 1 *A* boolean circuit with n inputs *is an acylic graph where the n inputs $x_1, \dots, x_n$ are part of the vertices. The internal vertices are labeled with $\wedge$, $\vee$ (with 2 incoming edges) or $\neg$ (with 1 incoming edge), with an additional distinguished vertex o that is the output (with no exiting edge). The size $|C|$ of a circuit C is its number of vertices (excluding the input ones). For a word $x \in \{0,1\}^*$, the notation $C(x)$ refers to the output of the circuit C if the input vertices of C are valued with the bits of x.*

Definition 2 *For a function $t : \mathbb{N} \to \mathbb{N}$, a* family of circuit of size $t(n)$ *is a sequence $(C_n)_{n\in\mathbb{N}}$ such that: C_n is an n-input circuit and $|C_n| \leq t(n)$.*

Definition 3 *A language $L \subseteq \{0,1\}^*$ is* decided by a family of circuit $(C_n)_{n\in\mathbb{N}}$ *if for all $n \in \mathbb{N}$, for all $w \in \{0,1\}^n$, we have: $C_n(w) = 1 \Leftrightarrow w \in L$.*

Definition 4 *For a function $t : \mathbb{N} \to \mathbb{N}$, we define $\mathsf{SIZE}(t) := \{L \subseteq \{0,1\}^* \mid L$ is decided by a family of circuits of size $O(t(n))\}$.*

Definition 5

$$\mathsf{P}/poly := \cup_{k\in\mathbb{N}} \mathsf{SIZE}(n^k)$$

1. Show that any language $L \subseteq \{0,1\}^*$ is in size $\mathsf{SIZE}(n \cdot 2^n)$.
2. Show that for all function $t(n) = 2^{o(n)}$, there exists $L \notin \mathsf{SIZE}(t(n))$.
3. Show that every unary language is in $\mathsf{P}/poly$.
4. Exhibit a undecidable language that is in $\mathsf{P}/poly$.
5. Show that $\mathsf{P}/poly$ is not countable.

Solution:

1. Let $L \subseteq \{0,1\}^*$. For all $n \in \mathbb{N}$, we define $f_n : \{0,1\}^n \to \{\ 0,1\}$ by $f_n(w) = 1 \Leftrightarrow w \in L$, for all $w \in \{0,1\}^n$. Now, let $n \in \mathbb{N}$. Let us construct C_n with $O(n \cdot 2^n)$ vertices such that $C_n(w) = f_n(w)$ for all $w \in \{0,1\}^n$. The function f_n can be represented as a two-column table with 2^n entries where each valuation of n variables to either 0 or 1 is associated 0 or 1. This table can represented as a DNF $\phi = \vee_{1\leq j\leq k}(\wedge_{1\leq i\leq n} x_i = w_i^j)$ where $(w^j)_{1\leq j\leq k}$ (for some $k \leq 2^n$) are the words of $\{0,1\}^n$ ensuring $f_n(w_i) = 1$. Each clause $(\wedge_{1\leq i\leq n} x_i = w_i^j)$ can be represented by a circuit with $O(n)$ vertices. As there are at most 2^n of them, the formula ϕ can be represented by circuit of size $O(n \cdot 2^n)$.

2. Let us find an upper bound on the number of circuits $d(n)$ of size $t(n)$. There are at most $t(n)$ internal vertices, each labeled by either $\vee$, $\wedge$, or $\neg$. Furthermore, each vertex has at most two predecessors taken among $n + t(n)$ vertices. Overall, we have:

$$d(n) \leq 3^{t(n)} \cdot ((t(n)+n)^2)^{t(n)} = (3 \cdot (t(n)+n)^2)^{t(n)} = 2^{t(n)\log((3\cdot(t(n)+n)^2))}$$

In addition, there are 2^{2^n} functions from $\{0,1\}^n \to \{0,1\}$. Since $t(n) = 2^{o(n)}$, we have $t(n) \cdot \log((3 \cdot (t(n)+n)^2)) = o(2^n)$. Thus $d(n) = 2^{o(2^n)}$. It follows that, asymptotically, there is not enough circuits of size $t(n)$ to compute all Boolean functions.[1]

3. Consider a unary language $L \subseteq 1^*$. For $n \in \mathbb{N}$ we build the circuit C_n such that, if $1^n \in L$, then C_n consists of $\wedge$ vertices leading to the output, whereas if $1^n \notin L$, we consider a circuit C_n that always yields false (for instance, by having $x \wedge \neg x$ for some input x). Then, for all n, we have $|C_n| = O(n)$ and $C_n(w) = 1 \Leftrightarrow w \in L$.

4. The language

   $L = \{\ 1^n \mid$ the binary encoding of n encodes a Turing machine in that always stops$\}$

   is unary and undecidable.

5. There exists a bijection between the set of unary languages and the set of subsets $\mathcal{P}(\mathbb{N})$ of $\mathbb{N}$ (which associates to a unary language $L \subseteq 1^*$ the set of $n \in \mathbb{N}$ such that $1^n \in L$). Since $\mathcal{P}(\mathbb{N})$ is not countable, so is $\mathsf{P}/poly$.

Exercise 2 Some alternation

1. Exhibit a polynomial time alternating algorithm that solves QBF.

2. Let $\mathsf{ONE}-\mathsf{VAL}$ be the problem of deciding whether a boolean formula is satisfied by exactly one valuation. Show that $\mathsf{ONE}-\mathsf{VAL} \in \Sigma_2^p$.

3. A boolean formula is minimal if it has no equivalent shorter formula – where the length of the formula is the number of symbols it contains. Let $\mathsf{MIN}-\mathsf{FORMULA}$ be the problem of deciding whether a boolean formula is minimal. Show that $\mathsf{MIN}-\mathsf{FORMULA} \in \Pi_2^p$.

Solution:

1.
```
qbf(nu,phi):
        case(phi):
                - phi: propositional formula
                        return yes iff nu stisfies phi
                - phi = exists x, phi'
                        (exists) choose i in [0,1]
                        qbf(nu[x = i],phi')
                - phi = forall x, phi'
                        (forall) choose i in [0,1]
                        qbf(nu[x = i],phi')
```

[1] Question and solution inspired from Sebastiaan A. Terwijn Complexity theory course notes.

Here, the number of alternations is unbounded.

2. OneVal(phi):

```
        (exists) choose a valuation nu
        if (nu satisfies phi)
        then
                (forall) choose a valuation nu'
                if (nu' does not satisfy phi) or (nu = nu')
                then return TRUE
                else return FALSE
        else
                return FALSE
```

   Here we have one alternation, with first the existential states (exists) and then the universal states (forall).

3. MinFormula(phi):

```
        (forall) choose a formula psi with |psi| < |phi|
        (exists) choose a valuation nu
        if nu does not satisfy phi <-> psi
        then
                return TRUE
        else
                return FALSE
```

   Here we have one alternation, with first the universal states (forall) and then the existential states (exists).

Exercise 3 Collapse of PH

1. Prove that if $\Sigma_k^P = \Sigma_{k+1}^P$ for some $k \geq 0$ then $\mathsf{PH} = \Sigma_k^P$. (Remark that this is implied by $\mathsf{P} = \mathsf{NP}$).

2. Show that if $\Sigma_k^P = \Pi_k^P$ for some k then $\mathsf{PH} = \Sigma_k^P$ (*i.e.* PH collapses).

3. Show that if $\mathsf{PH} = \mathsf{PSPACE}$ then PH collapses.

4. Do you think there is a polynomial time procedure to convert any QBF formula into a QBF formula with at most 10 variables ?

Solution:

First, note that $\Sigma_k^P = \mathsf{co}\,\Pi_k^P$ for all $k \geq 0$. In the following, all quantifications are made with a polynomial bound on the size of the variables considered.

1. Let us assume that $\Sigma_k^P = \Sigma_{k+1}^P$ for some $k \geq 0$, we prove by induction that $\forall j \geq k, \Sigma_k^P = \Sigma_j^P$, This holds for $j = i$. Now, consider some $j > i$ and assume that $\Sigma_k^P = \ldots = \Sigma_{j-1}^P$. Let $L \in \Sigma_j^P$. There exists a language $B \in \mathsf{P}$ ensuring: $x \in L \Leftrightarrow \exists y_1,\ \forall y_2,\ \ldots,\ Q_j y_j,\ (x, y_1, \ldots, y_j) \in B$.

   Let $L' = \{(x, y_1) \mid |y_1| \leq p(|x|) \land \forall y_2,\ \ldots,\ Q_j y_j,\ (x, y_1, y_2, \ldots, y_j) \in B\}$ for some polynomial function p. We have $L' \in \Pi_{j-1}^P = \mathsf{co}\ \Sigma_{j-1}^P = \mathsf{co}\ \Sigma_k^P = \Pi_k^P$. That is, $x \in L \Leftrightarrow \exists y_1,\ (x, y_1) \in L'$ with $L' \in \Pi_k^P$. In fact, $L \in \Sigma_{k+1}^P = \Sigma_k^P$ by hypothesis.

2. With the previous question, we just have to prove that $\Sigma_k^P = \Sigma_{k+1}^P$.

   Let $L \in \Sigma_{k+1}^P$. As previously, There exists a language $B \in \mathsf{P}$ ensuring: $x \in L \Leftrightarrow \exists y_1,\ \forall y_2,\ \ldots,\ Q_{k+1}y_{k+1},\ (x, y_1, \ldots, y_{k+1}) \in B$.

   We define $L' = \{(x, y_1) \mid |y_1| \le p(|x|) \wedge \forall y_2,\ \ldots,\ Q_{k+1}y_{k+1},\ (x, y_1, y_2, \ldots, y_{k+1}) \in B\}$ for some polynomial function p. We have $L' \in \Pi_k^P = \Sigma_k^P$ by hypothesis.

   That is, there exists $B' \in \mathsf{P}$ such that $x \in L' \Leftrightarrow \exists y_1,\ \forall y_2,\ \ldots,\ Q_k y_k,\ (x, y_1, \ldots, y_k) \in B'$. But then, we have $x \in L \Leftrightarrow \exists y,\ (x, y) \in L'$. This is equivalent to $x \in L \Leftrightarrow \exists y, \exists y_1,\ \forall y_2,\ \ldots,\ Q_k y_k,\ (x, y, y_1, \ldots, y_k) \in B'$. This can be rephrased as $x \in L \Leftrightarrow \exists y',\ \forall y_2,\ \ldots,\ Q_k y_k,\ (x, y', \ldots, y_k) \in B'$. It follows that $L \in \Sigma_k^P$.

3. If $\mathsf{PH} = \mathsf{PSPACE}$, then QBF is in Σ_k^P for some k. But QBF is a complete problem for PSPACE, and thus PH. Let there be $B \in \mathsf{PH}$, it can be reduced to $\mathsf{QBF} \in \Sigma_k^P$ in logspace, so $B \in \Sigma_k^P$. That is, $\mathsf{PH} = \Sigma_k^P$

4. It is unlikely that PH collapses, and the statement would imply the previous question.

Exercise 4 Oracles

Consider a language A. A Turing machine with oracle A is a Turing machine with a special additional read/write tape, called the oracle tape, and three special states: $q_{query}, q_{yes}, q_{no}$. Whenever the machine enters the state q_{query}, with some word w written on the oracle tape, it moves **in one step** to the state q_{yes} or q_{no} depending on whether $w \in A$.

We denote by P^A (resp. NP^A) the class of languages decided in by a deterministic (resp. non-deterministic) Turing machine running in polynomial time with oracle A. Given a complexity class $\mathcal{C}$, we define $\mathsf{P}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathsf{P}^A$ (and similarly for NP).

1. Prove that for any $\mathcal{C}$-complete language A (for logspace reductions), $\mathsf{P}^{\mathcal{C}} = \mathsf{P}^A$ and $\mathsf{NP}^{\mathcal{C}} = \mathsf{NP}^A$.

2. Show that for any language A, $\mathsf{P}^A = \mathsf{P}^{\bar{A}}$ and $\mathsf{NP}^A = \mathsf{NP}^{\bar{A}}$.

3. Prove that if $\mathsf{NP} = \mathsf{P}^{\mathsf{SAT}}$ then $\mathsf{NP} = \mathsf{coNP}$.

4. Show that there exists a language A such that $\mathsf{P}^A = \mathsf{NP}^A$.[2]

5. We define inductively the classes $\mathsf{NP}_0 = \mathsf{P}$ and $\mathsf{NP}_{k+1} = \mathsf{NP}^{\mathsf{NP}_k}$. Show that $\mathsf{NP}_k = \Sigma_k^p$ for all $k \ge 0$.

Solution:

1. We do the proof for NP. Obviously, we have $\mathsf{NP}^{\mathcal{C}} \supseteq \mathsf{NP}^A$. Now, $B \in \mathsf{NP}^{\mathcal{C}}$. There exists a non-deterministic Turing machine running in polynomial time deciding B with an oracle $C \in \mathcal{C}$. We also have a logspace (and hence polynomial time) reduction f such that: $x \in \mathcal{C} \Leftrightarrow f(x) \in A$ since A is hard for $\mathcal{C}$. We build the non-deterministic Turing machine N' that executes N while replacing a call $u \in C$? with a call $f(u) \in A$?. The Turing machine N' also runs in polynomial time and decides B with the oracle A. That is, $B \in \mathsf{NP}^A$.

[2] In fact, there also exists a language B such that $\mathsf{P}^B \neq \mathsf{NP}^B$, which does not prove that $\mathsf{P} \neq \mathsf{NP}$.

2. We simply have to swap the states q_{yes} and q_{no} in the computation.

3. $\mathsf{P}^{\mathsf{SAT}}$ is a deterministic class, so it is closed by complementation. Hence, if $\mathsf{NP} = \mathsf{P}^{\mathsf{SAT}}$, we have $\mathsf{coNP} = \mathsf{NP}$

4. Consider $A = \mathsf{QBF}$. By question 1, we have $\mathsf{P}^{\mathsf{QBF}} = \mathsf{P}^{\mathsf{PSPACE}}$ and $\mathsf{NP}^{\mathsf{QBF}} = \mathsf{NP}^{\mathsf{PSPACE}}$. Furthermore, $\mathsf{NP}^{\mathsf{PSPACE}} \subseteq \mathsf{NPSPACE}$ since one can simulate the calls to the oracle in polynomial space (as there is a polynomial number of calls). Therefore, $\mathsf{NP}^{\mathsf{PSPACE}} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{PSPACE} \subseteq \mathsf{P}^{\mathsf{PSPACE}}$.