Advanced Complexity

TD n°4

Charlie Jacomme

October 10, 2018

Exercise 1 : Language theory

Show that the following problems are PSPACE-complete :

1. NFA Universality :
   — INPUT : a non-deterministic automaton A over alphabet Σ
   — QUESTION : $\mathcal{L}(A) = \Sigma^*$?
   Bonus : what is the complexity of this problem for a DFA ?
2. NFA Equivalence
   — INPUT : two non-deterministic automata A_1 and A_2 over the same alphabet Σ
   — QUESTION : $L(A_1) = L(A_2)$
   Bonus : what is the complexity of this problem for a DFA ?
3. DFA Intersection Vacuity :
   — INPUT : deterministic automata $A_1, \dots, A_m$ for some m
   — QUESTION : $\bigcap_{i=1}^{m} L(A_i) = \emptyset$?

Solution:

1. The converse problem is in NP, and PSPACE is stable by complement (remark that there will be a word not accepted of polynomial size in the size of the automaton if there is a word not accepted). For the hardness, suppose M is a TM with polynomial space bound $p(n)$, and w is an input to M of length n. We will show how to take M and w, and write down, in polynomial time, a regular expression E that is Σ^* if and only if M does not accept w (PSPACE is closed under complementation).

   Considering that a run of M can be written by writing the sequence of configurations (of length $p(n)$ with the blank symbols) separated by a #.Then, we construct $E = F + G + H$ where F,G and H respectively define the runs that do not start right, move right, or finish right :
   — H : finishes wrong. M fails to accept if the sequence has no accepting state. Thus, let $H = (\Sigma - Q_f)^*$, where Q_f is the set of accepting states of M.
   — F : Starts wrong. Any string in which the first $p(n) + 2$ symbols are not #, q_0 (the start state), w, and $p(n) - n$ blanks, is not the beginning of an accepting computation, and so should be in $L(E)$. We can write F as the sum of the terms :
      — $(\Sigma - \{\#\})\Sigma^*$, i.e., all strings that do not begin with #.
      — $\Sigma(\Sigma - q_0)\Sigma^*$, i.e., all strings that do not have q_0 as their second symbol.
      — $\Sigma^{i+1}(\Sigma - a_i)\Sigma^*$, where a_i is the ith position of w.
      — $\Sigma^i(\Sigma - B)\Sigma^*$, for all $n + 3 \leq i \leq p(n) + 1$.
      — $(\Sigma + \epsilon)^{p(n)+1}$. This term covers all strings that are shorter than $p(n) + 2$ symbols, and therefore cannot have an initial state, regardless of the symbols found there.
   — G : moves wrong. We need to capture all strings that have some point at which symbols separated by distance roughly $p(n)$ do not reflect a move of M. The idea is similar to that used in Cook's theorem . Each position of a configuration is determined by the symbol at that position in the previous configuration and the

two neighboring positions. Thus, G is the sum of terms $(\Sigma^*)UVW(\Sigma^{p(n)})X(\Sigma^*)$, where U, V, W, X are four symbols of Σ such that if UVW were three consecutive symbols of a configuration of M, then X would not be the symbol in the same position as V in the next ID. For example, if none of U, V, W are a state, then X could be any symbol but V.
We can indeed compute those formulae in polynomial time, and if M accepts, $E \neq \Sigma^*$, and if M rejects, there is no accepting run and $E = \Sigma^*$

Bonus : For a DFA, we check if a non acceptable state can be reached, it is NL.

2. We can guess a word (letter by letter, because its size is bounded by $2^{|Q_1|+|Q_2|}$ which is accepted by one and not the other, we use the power set construction on the fly for the second one, and so the co problem is NP and the problem is PSPACE. We can also build the two minimal automata by exhaustive search and check if they are indeed equal, which is also PSPACE. For the hardness, we are given an automaton $\mathcal{A}$. We simply construct $\mathcal{B}$ such that $L(\mathcal{B}) = \Sigma^*$, and then :

$$L(\mathcal{A}) = \Sigma^* \Leftrightarrow L(\mathcal{A}) = L(\mathcal{B})$$

Bonus : For a DFA, the minimization is in $O(n \times log(log(n)))$ where n is the number of state with the Hopcroft algorithm.

3. As for the previous one we guess a word which invalidate the problem. For the hardness we are given $\mathcal{A}$. For the hardness, we can use the same proof as in the first one, but instead of making a global E, we build all the corresponding deterministic $\mathcal{A}_i$ and check the universality of the union. Beware, for the moves wrong regexp, we can easily produce a NFA detecting a wrong formation, but with determinism, we need $p(n)$ DFA, one for each position where the mistake might be.

Exercise 2 : Did you get padding ?

Show that if P = PSPACE, then EXPTIME = EXPSPACE.

Solution:

We assume that PSPACE $\subset P$. Let L_1 be accepted by $M_1 \in$ EXPSPACE, with M_1 deciding in $p(2^n)$ space, where p is a polynomial function. We define

$$L_2 = \{(x, 1^{2^{|x|}}) | M_1 \textit{ accepts } x \textit{ in space } p(|x| + 2^{|x|})\}$$

L_2 is accepted by M_2 which simulates M_1 and rejects if it uses too much space. Clearly, M_2 is in PSPACE $\subseteq P$. So we have M_3 which accepts L_2 in polynomial time. We can then build M_4 which on input x produces $(x, 1^{2^{|x|}})$ and simulates M_3. M_4 recognizes L_1, and $M_4 \in$ EXPTIME. We do have EXPSPACE $\subseteq$ EXPSPACE.

Exercise 3 : Too fast !

Show that ATIME($log\ n$) $\neq L$.

Solution:

When considering ATIME($log\ n$), we do not even have the time to read the full input. So any language which is in L and needs for the input to be completely read will yield the result. For instance, one may use the palindromes language, or 0^n on a two letter alphabet, or 0^{2k} on a one letter alphabet.

Exercise 4 : Direct application

Show that EXPSPACE = AEXPTIME.

Hint : You may use that if f is space-constructible, then :

$$\text{SPACE}(poly(f(n)) = \text{ATIME}(poly(f(n)))$$

Solution:
We set $f(n) = 2^n$, then :

$$\mathsf{SPACE}(2^{p(n)}) \subseteq \mathsf{SPACE}(poly(2^{p(n)}) = \mathsf{ATIME}(poly(2^{p(n)})) \subseteq \mathsf{AEXPTIME}$$

As p is arbitrary, then $\mathsf{EXPSPACE} \subseteq \mathsf{AEXPTIME}$. And conversely :

$$\mathsf{ATIME}(2^{p(n)}) \subseteq \mathsf{ATIME}(poly(2^{p(n)}) = \mathsf{SPACE}(poly(2^{p(n)})) \subseteq \mathsf{EXPSPACE}$$

Exercise 5 : Closure under morphisms

Given a finite alphabet Σ, a function $f : \Sigma^* \to \Sigma^*$ is a morphism if $f(\Sigma) \subseteq \Sigma$ and for all $a = a_1 \cdots a_n \in \Sigma^*$, $f(a) = f(a_1) \cdots f(a_n)$ (f is uniquely determined by the value it takes on Σ).

1. Show that NP is closed under morphisms, that is : for any language $L \in \mathsf{NP}$, and any morphism f on the alphabet of L, $f(L) \in \mathsf{NP}$.
2. Show that if P is closed under morphisms, then $\mathsf{P} = \mathsf{NP}$.

Solution:

1. If we are given L and M a NTM recognizing L, then we can build M' which on inputs $f(a)$ guesses $a_1, ..., a_n \in \Sigma^n$ such that $f(a) = f(a_1)...f(a_n)$ and finally simulates M on $a_1...a_n$. M' is a NTM in polynomial time which recognizes $f(L)$. Thus $f(L) \in \mathsf{NP}$.
2. SAT is in NP, and in particular, with the certificate definition, we know that

   $$L = \{(\phi, u) | \text{u is a correct valuation of } \phi\}$$

   is in P. We may consider an alphabet such that the symbols for ϕ and u are disjoints, and then, we may consider the morphism f such that $f(\phi) = \phi$ and $f(u) = 0^{|n|}$. We have $f(L) = SAT$, $L \in P$, and by hypothesis, P is closed under morphism. Thus, $SAT \in P$, and with the Cook-Levin theorem, $\mathsf{P} = \mathsf{NP}$.

Exercise 6 : Unary Languages 1.

Prove that if a unary language is NP-complete, then $\mathsf{P} = \mathsf{NP}$.
Hint : consider a reduction from SAT to this unary language and exhibit a polynomial time recursive algorithm for SAT

2. Prove that if every unary language in NP is actually in P, then $\mathsf{EXP} = \mathsf{NEXP}$.

Solution:

1. Suppose we have a unary language U NP-complete. We then have a reduction R from SAT to U. $R(\phi)$ is computed in polynomial time, so we have p such that $|R(\phi)| \leq p(|\phi|)$. Basically, we can then use the self reducibility of SAT, but by cutting some recursions branching by using the fact that $R(\phi) = R(\psi)$ if and only if ϕ and ψ are both satisfiable or both un-satisfiable. We will write $\phi(t)$ where $t \in \{0,1\}^*$ to consider partial evaluation of ϕ where we substituted x_i with the truth value of t_i. This yields the algorithm, where n is the number of variables in ϕ :

```
Initialise hash table H
Sat(φ)
  if  |t| = n then return 'yes' if φ(t) has no clauses,
                                 else return 'no'
  Otherwise, if R(φ(t)) ∈ H, then return H(R(φ(t)))
  Otherwise, return 'yes' if either Sat(φ(t0)) or Sat(φ(t1)).
```

```
        return no otherwise
      In both case, set H(R(φ(t)) to the answer
```

There will be at most $p(n)$ different possibles values for the $R(\phi(t))$ (U is unary), so there will be at most $p(n)$ recursive call of the functions. And in every recursive call, we make a computation of R in time $p(n)$. So our algorithms runs in $O(p^2(n))$ wich is in P. Thus $SAT \in P$, and $\mathsf{P} = \mathsf{NP}$.

2. For a language L decided in time $T(n)$, we define $L_{pad} = \{1^{(x,10^{T(|x|)})}, x \in L\}$. Let $L \in NEXP$ recognized by N in time $T(n)$ exponential. We build $N' \in \mathsf{NP}$ which recognizes L_{pad} :
   — On input 1^m, check the well-formdness to obtain $(x, 10^y) = m$
   — Simulate N on x for at most y step
   — Either return the result of N , or reject in case of time out.
   N' does recognizes L_{pad}, and it runs in polynomial times for the first step, and then y step for the second, with y being part of the input. Thus, $N' \in \mathsf{NP}$. But then by assumption, $L \in P$, and we have M a DTM which recognizes L_{pad} in polynomial time. We thus simply construct M' which is in exponential time, which given x computes $1^{(x,10^{T(|x|)})}$ and then simulate M with this input, and we are done.