Complexité avancée - TD 4

Simon Halfon

October 4, 2016

Exercise 1: Alternating Turing machines with negations

Let us define an alternating Turing machine with negations as a Turing machine where the set of non-halting states is partitioned into the set of *existential states*, the set of *universal states* and the set of *negation states*. Moreover there is the restriction that each configuration on a negation state has exactly one successor configuration. Remark that we do not require that the machine always halts.

For such a machine $\mathcal{M}$ we define the set of *eventually accepting* configurations, and the set of *eventually rejecting* configurations as the minimal sets of configurations satisfying the following conditions:

- if C is an accepting configuration, then C is *eventually accepting*;
- if C is an existential configuration and there exists a successor configuration C' of C (i.e, $C \rightarrow_{\mathcal{M}} C'$) which is *eventually accepting*, then C is *eventually accepting*;
- if C is a universal configuration, and all successor configurations C' of C are *eventually accepting*, then C is *eventually accepting*;
- if C is a negation configuration and the (unique) successor configuration C' of C is *eventually rejecting*, then C is *eventually accepting*;
- if C is a rejecting configuration, then C is *eventually rejecting*;
- if C is an existential configuration and all successor configuration C' of C are *eventually rejecting*, then C *is eventually rejecting*;
- If C is universal configuration, and there exists a successor configuration C' of C which is *eventually rejecting*, then C is *eventually rejecting*;
- if C is a negation configuration and the (unique) successor configuration C' of C is *eventually accepting*, then C is *eventually rejecting.*

The machine accepts an input x iff the initial configuration on input x is *eventually accepting.* The language accepted by an alternating Turing machine with negations $\mathcal{M}$ is the set of all x accepted by $\mathcal{M}$.

Prove that any alternating Turing machine $\mathcal{M}$ with negations can be simulated by an alternating Turing machine $\mathcal{M}^*$ without negations, with no extra cost in time or space. More precisely prove that there exists a configuration reachable in n steps and using m working tape cells in $\mathcal{M}$ iff there exists a configuration reachable in n steps and using m working tape units in $\mathcal{M}^*$. Do not assume any space or time bound on $\mathcal{M}$.

Exercise 2: Alternating logarithmic time vs logarithmic space
Show that $\mathsf{ATIME}(log\ n) \neq \mathsf{L}$.
Hint: show that the language of palindromes is in one class but not the other

Exercise 3: Yet another padding argument
Show that $\mathsf{EXPSPACE} = \mathsf{AEXPTIME}$. (skip this exercise if you have seen it in class)

Exercise 4: Linearly and logarithmically bounded alternations
Let $\mathsf{AP}(O(n))$ (resp. $\mathsf{AP}(O(log\ n))$) be the class of problems which can be decided by an alternating polynomial time Turing machine whose computations have a linear (resp. logarithmic) number of alternations (in the size of the input).

- Is QBF in $\mathsf{AP}(O(n))$? In $\mathsf{AP}(O(log\ n))$?
- Can we conclude $\mathsf{PSPACE} = \mathsf{AP}(O(n))$? $\mathsf{PSPACE} = \mathsf{AP}(O(log\ n))$?

Exercise 5: P-complete problems Show the following problems to be P-complete:

- – INPUT: G a context-free gramar
  – QUESTION: $\mathcal{L}(G) = \emptyset$?
- – INPUT: G a context-free grammar, and w a word
  – QUESTION: $w \in \mathcal{L}(G)$?

  Hint: use the Chomsky Normal Form: given a grammar, one can compute in logarithmic space an equivalent grammar with production rule of the form $X \to YZ; X \to a$ or $S \to \varepsilon$.
- – INPUT: A *planar* circuit $\mathcal{C}$
  – QUESTION: Does $\mathcal{C}$ evaluate to true ?

Exercise 6: 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 $\mathsf{P} = \mathsf{NP}$.

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 $\mathsf{EXP} = \mathsf{NEXP}$.
   Hint: remember we can always restrict our attention to Turing machines on alphabet $\{0, 1\}$.
3. Show the converse.