Formal Methods
Fall 2017

S07

Professor : Ultes-Nitsche Ulrich
Assistant : Christophe Stammet

Submitted by Sylvain Julmy

Exercise 1

(1)

Cook's theorem is important for complexity theory because it it a proof about the $\mathcal{NP}$-completeness of the `SAT` algorithm. Usually, we demonstrate that a problem is in the $\mathcal{NP}$-complete class by reducing it in polynomial time to a problem which is already $\mathcal{NP}$-complete . The problem is to have a known $\mathcal{NP}$-complete problem to use and Cook demonstrate that `SAT` is $\mathcal{NP}$-complete .

(2)

In order to demonstrate that any problem can be reduce to `SAT`, we don't encode the problem itself but the algorithm that is solving it. We know there exist such an algorithm because we encode problem that are in $\mathcal{NP}$. ϕ represents the algorithm in propositional logic.
The variables we are using for the proof are the following :

- $X_{i,j}$ which represents the jth symbols after step i,
- $y_{i,j,z}$ which represents that's after step i, the square j contains the symbol z.

An interpretation of ϕ sets to true means that ϕ is an accepting computation of the non-deterministic Turing Machine M.

(3)

ϕ consist of 4 sub-formula $\phi = U \wedge S \wedge N \wedge F$ where

- U represents that each symbol in the ID sequence is unique, on other words, each square has exactly one symbol after each step.
- S represents that's the start ID is correct, which means that there exist only one unique initial state.
- N represents that's the next ID is correct, which means that after a step, the head of M is in exactly one position.
- F represents that M is ending in an accepting state, which is $X_{p(n),y}$ is set to *true* where q_y is the accepting state.

All of those formulae are necessary to validate that our initial configuration is well-defined.

(4)

We know that $p(n)$ is polynomial and the formula N is used to validate that the Turing Machine is in exactly one state after each step. We constuct the N formula like the following :

$$(N_0, N_1, \cdot, N_{p(n)})$$

where

$$N_i = (A_{i,0} \vee B_{i,0}) \wedge (A_{i,1} \vee B_{i,1}) \wedge \cdots \wedge (A_{i,p(n)} \vee B_{i,p(n)})$$

$A_{i,j}$ represents that the jth symbol in the ith id is the state and $B_{i,j}$ represents that the jth symbol in the ith id is not the state.
So it means that the Turing machine is in at least one state after each step and that it can't be in two states at once.
This construction is in $O(p(n)^2)$ which is polynomial, because $p(n)$ is polynomial.

(5)

For this exercice, we would use a slightly different notation in order to be more expressive about the formula itself :

- $Q_{i,j}$ represents that's after step i, M is in state q_j.
- $H_{i,j}$ represents that's the head of M is in the jth square after the step i.
- $S_{i,j,z}$ represents that's after step i, the square j contains symbol z. This is just some redefinition of $X_{i,j}$ and $y_{i,j,z}$.

The sentence "M starts on input "1001"" is is descrive by the following formulas :

$$\begin{aligned} S = &Q_{0,0} \wedge && \text{the initial state (at step 0) is } q_0 \\ &H_{0,1} \wedge && \text{the head is on the first square of the tape at step 0} \\ &S_{0,1,1} \wedge S_{0,2,0} \wedge S_{0,3,0} \wedge S_{0,4,1} \wedge && \text{the tapes contains 1 at square 1, 0 at square 2, ...} \\ &S_{0,1,1} \wedge S_{0,1,1} \wedge \cdots \wedge S_{0,p(n),B} && \text{the rest of the tape is containing blank symbol.} \end{aligned}$$

Exercise 2

The idea of Cook in order to prove that `SAT` is $\mathcal{NP}$-complete is to demonstrate that any algorithm that is solving a problem can be encode into a propositional formula in a polynomial time. We already know that `SAT` is $\mathbb{NP}$-Hard because a non-deterministic Turing Machine can solve `SAT` in a polynomial time. Then, if we show that any algorithm can be encoded as a propositional formula in a polynomial time, it would prove that `SAT` is $\mathcal{NP}$-complete .
To do so, we don't encode the problem itself but the Turing Machine that is solving the problem. We know that there exist a Turing Machine because we wan't to demonstrate $\mathcal{NP}$-completeness of known problem with algorithm.
Using the variables Q,H and S, we can describe any computation of a Turing machine in a polynomial size using the function $p(n)$. We know that $p(n)$ is polynomial because it runs on a non-deterministic Machine in polynomial time (we know the algorithm).

The proof also show that constucting the formulas used in order to represent the variables, configurations clauses and transitions clauses are in a polynomial complexity with respect to $p(n) : O(p(n)^{O(n)})$.
Finally, we have show that any algorithm are encodable into a propositional formula in complexity $O(p(n)^{O(n)})$. By any algorithm, there are also $\mathbb{NP}$-Hard ones then `SAT` is $\mathcal{NP}$-complete
.