SNEAKY COMPLETE LANGUAGES AND NOTES ON CHAPTER 7

JOEL FRIEDMAN

Contents

Disclaimer: The material may sketchy and/or contain errors, which I will elaborate upon and/or correct in class. For those not in CPSC 421/501: use this material at your own risk...

1. Where CPSC 421/501 Differs from Chapter 7 of [Sip]

Here we list a few minor differences between the definitions and proofs in [Sip], Chapter 7 from the way we cover them this year in CPSC 421/501.

[Sip] defines NP in terms of verifiers and then in Corollary 7.22 proves that

$$\mathrm{NP} = \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(n^k);$$

we define NP via the above equation.

If f, g are functions $\mathbb{N} \to \mathbb{R}$, we write $f(n) = O(g(n))$ if there are $C \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ such that $|f(n)| \le Cg(n)$ for all $n \ge n_0$. Hence $n^2 - 2n = n^2 + O(n)$. [Sip] insists that f, g have non-negative values; this doesn't create any problems when defining

$$\mathrm{P} = \bigcup_{k\in\mathbb{N}} \mathrm{TIME}(n^k),\ \mathrm{NP} = \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(n^k).$$

[Sip] proves the Cook-Levin Theorem by showing that a non-deterministic algorithm running in time n^k can be described by introducing Boolean variables x_{ijs} where $i, j \in n^k$ and $s \in Q\cup\Gamma\cup\{\#\}$; this uses the configuration notation on page 169 of [Sip] (e.g., $abqab$ for a tape with contents $abab$ followed by blanks, with the tape head over the third tape cell and the machine being in state q).

We use the more common type of notation, specifically:

$$x_{ij\gamma},\ y_{ij},\ z_{iq}$$

Date: Tuesday 14th November, 2023, at 09:15.

Research supported in part by an NSERC grant.

where i, j range over $1, 2, \ldots, Cn^k$, with i the step number and j the cell location, and $\gamma \in \Gamma$, and $q \in Q$. ([Sip] takes $C = 1$ for notational simplicity.) Here $x_{ij\gamma}$ is true iff at step i, the tape cell in position j has the value γ; y_{ij} is true iff the tape head is in cell position j on the i-th step; z_{iq} is true iff the machine is in state q on the i-th step.

2. Some Languages that are Complete for Sneaky Reasons

There is a standard way to produce languages that are complete for NP and PSPACE (under polynomial time reductions). The advantage is that the proofs of completeness are very simple; the disadvantage is that these languages aren't of practical interest. Let us start with the NP-complete language.

Let

$$\begin{aligned}&\text{NP-SNEAKY}\\ =&\{\langle M, w, 1^t\rangle \mid M \text{ is a non-deterministic T.m. that accepts } w \text{ within time } t\}.\end{aligned}$$

We claim that NP-SNEAKY is NP-complete. To prove this we need to show that (1) NP-SNEAKY lies in NP, and (2) any $L \in$ NP can be reduced in polynomial time to NP-SNEAKY. Claim (2) is almost immediate, and claim (1) requires a bit more thought: you run a (non-deterministic) universal Turing machine for t steps of M on input w, and you have to verify that the simulation runs in time polynomial of

$$\langle M\rangle + \langle w\rangle + t.$$

This is easy (since the input size is at least t), and will likely be done in class this year.

You should be aware that the simulation will *not* run in time polynomial of

$$\langle M\rangle + \langle w\rangle + \log_2 t.$$

Hence it is crucial that NP-SNEAKY expresses time, t, in *unary*, i.e., as 1^t. In other words, the language

$$\{\langle M, w, t\rangle \mid M \text{ is a non-deterministic T.m. that accepts } w \text{ within time } t\}$$

when you describe t in base 10 or in binary, is not in NP, at least not as far as we know.

You might compare this to showing that SAT is NP-complete: showing that SAT is in NP is easy, but the proof that any language in NP can be reduced to SAT is the essence of the Cook-Levin theorem, and is a much more elaborate proof. For NP-SNEAKY both steps in showing NP-completeness are easy, but showing that NP-SNEAKY lies in NP—which requires a universal Turning machine—is more difficult than that any languages in NP can be reduced to NP-SNEAKY.

Another comparison between NP-SNEAKY and SAT (and 3COLOR, VERTEX-EXPANSION, PARTITION, etc.) is that the latter problems are interesting in applications, whereas NP-SNEAKY is just a formal construction that doesn't seem to have applications beyond giving a language with a simple proof of NP-completeness.

Similar remarks hold for the language:

$$\begin{aligned}&\text{PSPACE-SNEAKY}\\ =&\{\langle M, w, 1^s\rangle \mid M \text{ is a deterministic T.m. that accepts } w \text{ using at most space } s\},\end{aligned}$$

which we easily show is complete for PSPACE under polynomial time reductions, i.e., (1) PSPACE-SNEAKY lies in PSPACE, and (2) if L lies in PSPACE, then there is a polynomial time reduction of L to PSPACE-SNEAKY. Moreover, PSPACE-SNEAKY is not as interesting as TQBF (totally quantified Boolean formulas that are true) or other languages in Section 8.2 of [Sip].

One can similarly show that

$$\begin{aligned}&\text{NPSPACE-SNEAKY}\\ =&\{\langle M, w, 1^s\rangle \mid M \text{ is a non-deterministic T.m. that accepts } w \text{ using at most space } s\}\end{aligned}$$

is complete for NPSPACE under polynomial time reductions. However, Savitch's Theorem implies that NPSPACE = PSPACE, so PSPACE-SNEAKY is also complete for NPSPACE under polynomial time reductions.

Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, CANADA.

E-mail address: `jf@cs.ubc.ca`

URL: `http://www.cs.ubc.ca/~jf`