# COMP90057 Advanced Theoretical Computer Science
## Complexity Workshop Questions
## Second (Spring) Semester 2020

**Tony Wirth and Xin Zhang**

All question numbers refer to Sipser textbook, third edition.

**Week 3 (17 Aug)**

- 3.11
- 7.1
- 7.7
- 7.10
- 7.11(a) - submit

**Week 4 (24 Aug)**

- 7.12
- 7.14
- 7.18
- 7.20 - submit

**Week 5 (31 Aug)**

- 7.22 - submit
- 7.24
- 7.26
- 7.29

**Week 6 (7 Sep)**

- 7.30 - submit
- 7.31
- 7.37
- 10.11

**Week 7 (14 Sep)**

- 7.46
- 10.19
- 10.22 - submit
- 8.4

**Week 8 (21 Sep)**

- 8.6 - submit
- 8.11
- 8.16

https://eduassistpro.github.io/
Assignment Project Exam Help
Assignment Project Exam Help
der
https://eduassistpro.github.io/
Add WeChat edu_assist_pro

# COMP90057 Advanced Theoretical Computer Science
Workshop test - Week 3
Semester 2, 2020
Xin Zhang[1]

## 1 Solution

Question 7.11 Let

$$EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}.$$

We provide a polynomial-time algorithm to verify the membership of $EQ_{\text{DFA}}$. Denote $L(A)$ and $L(B)$ by $L_A$ and $L_B$, respe ... at $L(C) = (\overline{L_A} \cap L_B) \cup (L_A \cap \overline{L_B})$. The constructi

It can be easily proven tha ... etric difference is $\varnothing$. We can decide $EQ_{\text{DFA}}$ ... ]

https://eduassistpro.github.io/

Since $E_{\text{DFA}}$ is in $P$, verifying $L(C) = \varnothing$ takes polynomial time as well. [*0.5 point*]

Assignment Project Exam Help

### Marker's comments

Assignment Project Exam Help
der

In this subject, I have often had students asking me the amount of *detail* they ought to provide in a solution, and what result ... textbook can be referred to ... lpful to refer to the result by name o ... d formally in lectures and w

https://eduassistpro.github.io/

In the setting of an exam or an assessment where limited time is giv ... you may want to first give an overview of the proof with only essential ste ... are made in the proof can be proved later either in details or with some broa ... course, the balance between mathematical rigorousness and linguistic conciseness is a fine one to strike. My advice is to guarantee first that you have effectively communicated your main idea and then go for proofs in greater granularity. For this question, the amount of detail presented in the example solution should suffice. For example, the construction of the DFA $C$ in the proof can be seen as a direct consequence of Theorem 1.25 (page 45, Sipser) and you can claim that $C$ can be defined in polynomial time without giving a proof. For completeness, a proof for this result is given nonetheless in Section 2. Since $E_{\text{DFA}} \in P$ is directly covered in the workshop, you should be able to use this fact without proof as well.

Add WeChat edu_assist_pro

## 2 Constructing DFAs

We include here a lemma that shows DFA $C$ can be constructed in polynomial time.

**Lemma 1.** *On input $\langle D_1, D_2\rangle$, configurations of two DFAs $D_1$ and $D_2$, one can construct DFA $D_\cup$, $D_\cap$ and $D_1'$, in polynomial time, such that*

- *(union)* $L(D_\cup) = L(D_1) \cup L(D_2)$
- *(intersection)* $L(D_\cap) = L(D_1) \cap L(D_1)$

[1] with a few edits from Tony Wirth

- *(complement)* $L(D'_1) = \overline{L(D_1)}$

*Proof.* Let $D_1 = (Q_1, \Sigma, \delta_1, q_{s,1}, F_1)$ and $D_2 = (Q_2, \Sigma, \delta_2, q_{s,2}, F_2)$. For the ease of presentation, we shall assume that the two DFAs share the same alphabet. If the assumption is false, we can replace their individual alphabets with the union of the two, while keeping everything else the same, and use the modified configurations as the new input. We prove case by case.

**Union** Define

$$D_\cup = (Q_1 \times Q_2, \Sigma, \delta_3, (q_{s,1}, q_{s,2}), F_\cup),$$

where $Q_1 \times Q_2$ is the Cartesian product of $Q_1$ and $Q_2$, $\delta_3$ is the combined transition function, and $F_\cup$ is the new accepting state set. Here, for all $q_1 \in Q_1$, $q_2 \in Q_2$ and $x \in \Sigma$, $\delta_3$ is defined as

$$\delta \quad (q_1, q_2), x \quad = \quad \delta_1(q_1, x), \delta_2(q_2, x) \quad .$$

Since the idea is to *run* the tw nds up with an accepting state then $D_\cup$ s $_1 \in F_1$ or $q_2 \in F_2$.

https://eduassistpro.github.io/

**Intersection** $D_\cap$ is constructed similarly as we also intend to run the input for both machines in parallel. The difference here is that we only accept when *both* machines accept. Define

Assignment Project Exam Help

$$D_\cap = (Q_1 \times Q_2, \Sigma, \delta_3, (q_{s,1}, q_{s,2}), F_\cap),$$

Assignment Project Exam Help
der

where $F_\cup = \{(q_1, q_2) \quad q \quad F$ and $q \quad F \quad .$

**Complement** The co to switch the accepting and rejectin

https://eduassistpro.github.io/

$$D'_1 = (Q_1, \Sigma, \delta_1, q_{s,}$$

Add WeChat edu_assist_pro

It is easy to verify that descriptions of the new DFAs are in size polynomial in the input size and thus take polynomial time to construct. □

# COMP90057 Advanced Theoretical Computer Science
# Solutions to selected workshop questions (2020-08-24)
# Second (Spring) Semester 2020

**Solutions by Tony Wirth and Xin Zhang**[1]

- Sipser 7.7 • Closure of NP

Let $A, B$ be two arbitrary languages in NP. We first show that NP is closed under union. We show that there is a polynomial-time verifier for the language $A \cup B$. To verify membership in $A \cup B$, we let the Turing machine run *alternating* steps of polynomial-time $B$, there is a certificate for which ll thus accept the input

https://eduassistpro.github.io/

There is a polynomial-time overhead for this simulation, incorporating the fact that the size of the two configurations is polynomial in the input size. In particular, we could run this on two tapes: each verifier runs in polynomial time, so the running time is dominated by the sum of the polynomials. There is (potentially) quadratic overhead to simulate on one tape.

Assignment Project Exam Help
Assignment Project Exam Help
der

We now show that NP is closed under concatenation. We show that there is a polynomi ership in $A \circ B$, the certific A component of the input a e indicates where the input is split between the two languages steps of the two verifiers, until both halt. The total run the larger of the inputs, and hence in the total input, as t in the size of the two configurations.

https://eduassistpro.github.io/
Add WeChat edu_assist_pro

- Sipser 7.10 • $ALL_{\mathrm{DFA}}$

It is quick to decide whether the input is well formed. To test whether an DFA $D$ is in $ALL_{\mathrm{DFA}}$, construct the automaton $D'$ whose accept and reject states have been flipped. This is a polynomial-time operation, and the size of $D'$ is polynomial in the size of $D$. Then carry out the marking process of the proof that $E_{\mathrm{DFA}}$ is a decidable language (Theorem 4.4). Each repetition marks some state: To mark a state requires scanning every transition, then passing through the whole input to write to the tape. Potentially, the running time might be proportional to the cube of the size of the input, but no more. Since $D$ is a DFA, the DFA $D'$ accepts the complement language to $L(D)$, hence $D$ is in $ALL_{\mathrm{DFA}}$ if and only if $D'$ is in $E_{\mathrm{DFA}}$.

[1] thanks to Jess McClintock

- Sipser 7.14 • *PERM POWER*

The first-thought approach to decide *PERM POWER* would be to apply the permutation $q$ to itself $t-1$ times and then check for equality with $p$. To apply a permutation to itself requires something like $k^2 \log k$ running time, so the overall running time is roughly $\Theta(tk^2 \log k)$. However, the *size* of the input is actually only $O(k \log k + \log t)$, as we represent $t$ with $O(\log t)$ bits, while $O(\log k)$ bits represent each member of $\{1, 2, \ldots, k\}$. So, this first approach has running time that is exponential in the input size, in relation to $t$.

The trick is to determine $q, q^2, q^4, q^8, \ldots$ up to and including $q$ permuted with itself to largest power of two less than $t$. For example, to determine $q^{10}$, we can calculate $((q$ ds, permutations compo have at most $\log t$ perm ns to apply. the binary representation of $t$ governs exactly which. Hence the running time is $O(k^2(\log k)(\log t))$, which is polynomial in the input size.

https://eduassistpro.github.io/
Assignment Project Exam Help
Assignment Project Exam Help
der
https://eduassistpro.github.io/
Add WeChat edu_assist_pro

COMP90057 Advanced Theoretical Computer Science
Workshop assessed question – Week 4
Semester 2, 2020
Xin Zhang and Tony Wirth

# 1 Solution

*PATH* is a relatively easy problem to solve in P, as the complexity is simply $O(m)$. Here $m$ is the number of edges in the input graph. It is hard to believe that an NP problem (language) can be reduced in polynomial time to a problem (language) that admits a linear-time algorithm.

From Problem 7.18 if P = NP, then *PATH*, a P language, must be NP-complete. The contrapositive form of the above statement is: if *PATH* / NP-complete, then P $\neq$ NP.

A proof of Problem 7.18 is also i

https://eduassistpro.github.io/

# 2 Marker's comments

Assignment Project Exam Help

This question has two parts, one to explain a general belief and the other to prove a formal statement. Only the second part contributes towards the marks. We provide an explanation of the general believe as to why *PATH* is not NP-complete, but any reasonable explanation would be fine.

Assignment Project Exam Help

der

Since 7.18 is **fully covered** in the workshop, if the submission relies on the result from 7.18 then it receives full marks s basically the same key points. The fo

https://eduassistpro.github.io/

**Direct Proof** Since *PATH* $\in$ P, if *PATH* $\notin$ NP-complete, then there must exist some NP language $L$ such that $L$ is not polynomial-time reducible to $L \notin$ P. Suppose for the sake of contradiction that $L \in$ P. Then we can define a pol $f$ from $L$ to *PATH*. On input $x$, $f$ runs a polynomial-time determin etermining whether or not $x$ is in $L$. The existence of such a decider follows from $L \in$ P. If $x \in L$, then $f$ returns an instance $y \in$ *PATH*. If $x \notin L$, then $f$ returns an instance $y' \notin$ *PATH*. Obviously, since *PATH* is a non-trivial language, such $y$ and $y'$ must exist. The strings $y$ and $y'$ are *hard coded*, and the running time of this part of the reduction is independent of $|x|$, once the $x \in L$ question has been answered. But this contradicts to the fact that $L$ is not polynomial time reducible to *PATH*. The claim must be true, and thus P $\neq$ NP.

Add WeChat edu_assist_pro

**Observations** When marking a submission that follows a direct approach, I am looking for

- A reflection on the statement asked by the question [0.5 points]
- Showing that a specific language that is in NP but not in P [1 point]

Many have made an attempt to show such a language only in P, but either not provided proof or have an incorrect proof. Some claimed that for all language $A, B \in$ P, $A \leq_p B$. This is not true if $B$ is trivial, i.e., either $\varnothing$ or $\Sigma^*$. Recall that in the definition of polynomial-time reduction, the reduction function needs to map a member of $A$ to a member of $B$ and a non-member of $A$ to a non-member of $B$. The former is not possible if $B$ is $\varnothing$ while the latter is not possible if $B$ is $\Sigma^*$.

# 3 Solution to 7.18

Consider an arbitrary language $A \in \mathrm{P}$, except for $\varnothing$ and $\Sigma^*$. Since $\mathrm{P} = \mathrm{NP}$, $A$ is also in NP. Suppose $B$ is an arbitrary language in NP and thus by the problem statement $\in \mathrm{P}$, we construct a polynomial-time reduction from $B$ to $A$ as follows. On an input $x$, we decide whether or not $x$ is in $B$. If $x \in B$, then produce a string $y_1$ that is in $A$; otherwise, if $x \notin B$, produce a string $y_2$ that is not in $A$. Since these strings $y_1$ and $y_2$ do not depend on the size of the input $x$ (once membership in $B$ has been determined), the running time of this process is polynomial in $|x|$ due to $B \in \mathrm{P}$.

Since $A$ is neither $\varnothing$ nor $\Sigma^*$, there always exists such $y_1$ and $y_2$. Since $A \in \mathrm{NP}$ and $B \leq_{\mathrm{P}} A$, $A$ is NP-complete.

https://eduassistpro.github.io/

Assignment Project Exam Help
Assignment Project Exam Help
der
https://eduassistpro.github.io/
Add WeChat edu_assist_pro

August 31, 2020

COMP90057 Advanced Theoretical Computer Science
Workshop assessed question – Week 5
Semester 2, 2020
Xin Zhang and Tony Wirth

# 1 Solution

A certificate for *DOUBLE-SAT* is simply two (distinct) satisfying assignments. Since each can be verified in polynomial time, and checked for distinctness, *DOUBLE-SAT* is in NP.

We reduce *SAT* to *DOUBLE-SAT* in polynomial time thus. Given an instance $\psi$ of *SAT*, we consider two variables $x$, $y$ that do not appear in $\psi$ and create the formula $\phi = \psi \wedge (x \vee y)$. This takes polynomial time to determine and construct. Now if $\psi$ *SAT* then $\phi$ has at least two satisfying assignments (in "addition" to the s rue $y$ false, and the second being $x$ false $y$ true. T $\notin$ *SAT*, then $\phi$ is also unsatisfiable, so it is not in

https://eduassistpro.github.io/

**Alternative Proof:** A polynomial-time verifier can be devised by checking if a given truth assignment, which serves as a certificate, is satisfying. Problem *SAT* is polynomial-time reducible to *DOUBLE-SAT*: for a *SAT* instance $\langle\phi\rangle$, conjoin $\phi$ with a tautology[1] whose variables are not in $\langle\phi\rangle$. For example, one can use $(y \vee \bar{y})$ where $y$ is a variable not in $\langle\phi\rangle$.

Assignment Project Exam Help
Assignment Project Exam Help
der

# 2 Comments

https://eduassistpro.github.io/

Most submissions provided a good solution to this week's problem: well done! The marking scheme is

Add WeChat edu_assist_pro

- Prove *DOUBLE-SAT* is in NP [0.5 point]
- Give a reasonable reduction function for the NP-hardness proof [0.5 point]
- Prove the polynomial time reduction by verifying the three properties (in the definition) [0.5 point]

Specifically, you need to show that $\phi \in SAT$ if and only if $f(\phi) \in DOUBLE\text{-}SAT$, for your reduction function $f$, *and* show that $f$ is a polynomial-time function.

*September 3, 2020*

[1] A tautology is a Boolean formula that is always evaluated to be true regardless the truth assignment to the variables

COMP90057 Advanced Theoretical Computer Science
Workshop assessed question – Week 6, Q7.30
Semester 2, 2020
Xin Zhang and Tony Wirth

# 1 Solution

A coloring of the items is a certificate that is easily checked for validity.

The polynomial-time reduction from $\neq SAT$ (Problem 7.26) is this. For each variable $x$ in $\phi$, construct two elements of $S$, one for $x$ and another for $\neg x$. For each clause in $\phi$ add a set to $C$ comprising its literals, which by design are elements of $S$. Finally, for each variable $x$, add set $\{x, \neg x\}$ to $C$. This mapping can be executed in time polynomial in the size of $\phi$.

Consider a mapping of TRUE t ... $\neq$-assignment, each clause has either two REDs and one BL ... sponding to $\{x, \neg x\}$ have one ... $\langle S, C\rangle$ is in *SET-SPLITTING*.

https://eduassistpro.github.io/

If $\langle S, C\rangle \in$ *SET-SPLITTING*, then the $\{x, \neg x\}$ sets represent a consistent truth assignment. Moreover, each clause-related subset has either two TRUE one FALSE or vice versa, which leads to a valie $\neq$-assignment.

Assignment Project Exam Help
Assignment Project Exam Help
der

**Alternative Reduction:** Alternatively, one can show that *SAT* is polynomial-time reducible to *SET-SPLITTING*. The re ... t two elements of $S$, one for $x$ and another for ... riable in $\phi$. Now, for each clause in $\phi$, assu ... $l_2, \ldots, l_k, y\}$ to $C$. Finally, for each variable $x$ in $\phi$, add set $\{x, \neg x\}$ to $C$, and for the s ... lso add set $\{y, \neg y\}$ to $C$.

https://eduassistpro.github.io/

If $\phi$ has a satisfying assignment, then we can produce a valid co ... RED and FALSE to BLUE. The special variable $y$ is ass ... olored BLUE and $\neg y$ colored RED. This guarantees that each set in $C$ contains both BLUE and RED elements.

Add WeChat edu_assist_pro

If $\langle S, C\rangle \in$ *SET-SPLITTING*, then the $\{x, \neg x\}$ sets represent a consistent truth assignment. Particularly, if $y$ is colored BLUE, then RED literals are assigned TRUE, and if $y$ is colored RED, then BLUE literals are assigned TRUE. Each clause-related set has at least one literal evaluated to TRUE.

# 2 Comments

This week we further cement our understanding of NP-complete proofs. Reducing from $\neq SAT$ is the more straightforward approach of the two, but working with *SAT* directly also yields a clean solution. The marking scheme is simple

- Prove *SET-SPLITTING* is in NP. [0.5 point]
- Prove *SET-SPLITTING* is in NP-hard via a polynomial-time reduction [1 point]

Similar to previous submissions, small errors or typos that do not hinder the comprehension of the solution are tolerated and do not result in point loss. A common mistake is to not include the variable-related sets when defining $C$. This is easily overlooked, but without these sets, the truth assignment derived from a coloring might not be consistent. For instance, $x$ and $\neg x$ might be colored the same. Some students also struggled in identifying the reduction itself. You need to construct *both* $S$ and $C$ explicitly; neither of them should be assumed.

Most of this week's submissions are well thought-out, and carefully written. Everyone has put a lot of effort into them. Well done!

https://eduassistpro.github.io/

Assignment Project Exam Help
Assignment Project Exam Help
der
https://eduassistpro.github.io/
Add WeChat edu_assist_pro

*September 10, 2020*

COMP90057 Advanced Theoretical Computer Science
Workshop assessed question – Week 7, Q10.22
Semester 2, 2020
Xin Zhang and Tony Wirth

# 1 Solution

The discussion of the simulation of space-bounded nondeterministic machine, and configurations, applies here. If the space used by a nondeterministic machine is in $O(\log n)$, where $n$ is the size of the input, then the number of configurations is $n2^{O(\log n)}$, which is polynomial in $n$. This consideration can be used to show that NL $\subseteq$ P. Here we rely on it to conclude that BPL $\subseteq$ P.

Since the BPL machine is a decider, every branch of the machine halts and thus it cannot repeat a configuration. Hence the graph of t
start configuration – is a DAG. Via a
reach each configuration. Since th
of the number of paths is constant time per node (due to the constant-size transition function), this dynamic program is *solved* in time polynomial in the size of the graph. We can therefore simulate the behavior of the BPL machine with a polynomial-time machine, accepting if the proportion of computational paths reaching the accept configuration (assume there is just one) is at least $2/3$.

https://eduassistpro.github.io/
Assignment Project Exam Help
Assignment Project Exam Help
der

# 2 Comment

In this submission, the ma

https://eduassistpro.github.io/
Add WeChat edu_assist_pro

- Demonstrate the idea of simulation, including terminatio pt and reject [0.5 point]
- Successfully define the configuration graph [0.5 point]
- Establish the polynomial time complexity of the algorithm [0.5 point]

An important idea in the complexity part of the subject, especially the space complexity component, is computing on configuration graphs, to simulate the behavior of a (nondeterministic) Turing machine. We have seen it in action multiple times, including Cook–Levin theorem ($SAT \in$ NP-complete), and Savitch's theorem. Another key aspect of this question is to translate the acceptance probability of a TM to the proportion of paths leading to an accepting configuration. A common mistake is to think that *all* computational steps of a probabilistic Turing machine are probabilistic. A PTM can take deterministic steps, just as a nondeterministic TM can take a deterministic step in its computation.

COMP90057 Advanced Theoretical Computer Science
Workshop assessed question – Week 8, Q8.6
Semester 2, 2020

Xin Zhang and Tony Wirth

# 1 Solution

For every language $A \in$ PSPACE-hard, we aim to show that $A$ is also NP-hard. For every language $B \in$ NP, language $B$ is also in PSPACE as NP $\subseteq$ PSPACE. It follows from the definition of PSPACE-hard that $B \leq_{\mathrm{P}} A$, and thus $A$ is NP-hard.

# 2 Comment

Following last week's submis
relief. For the first time, all the submi

https://eduassistpro.github.io/

There is not much to say about the question itself. We simply apply the definition of NP-hardness and PSPACE-hardness. The result NP $\subseteq$ PSPACE can be easily proved. In the textbook (p. 336), this is shown through the fact that NP $\subseteq$ NPSPACE = PSPACE, where the last equality follows from Savitch's theorem.

Assignment Project Exam Help
Assignment Project Exam Help
der
https://eduassistpro.github.io/
Add WeChat edu_assist_pro

*September 24, 2020*