Computer Science 511

$\mathcal{NP}$-Completeness: Classical Reductions

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #12

Goal for Today

Goals for Today:

- Review of a process to prove that a given language is $\mathcal{NP}$-complete
- Application of this to prove the $\mathcal{NP}$-completeness of several languages
- Saying a bit more about $\mathcal{NP}$-complete problems and sources of information about this

$\mathcal{NP}$-Completeness

- Recall that (we are saying that) a language L is ***$\mathcal{NP}$-hard*** (respectively, ***$\mathcal{NP}$-complete***) it L is hard (respectively, complete) for the complexity class $\mathcal{NP}$ with respect to ***polynomial-time many-one reductions***.
- See Lectures #6, #7 and #10 for the definitions of polynomial-time many-one reductions, hardness and completeness.
- At this point in the course, ***only one*** reasonably “natural” $\mathcal{NP}$-complete language — L_{FSAT}, which consists of encodings of satisfiable Boolean formulas — has been identified. The fact that this language is $\mathcal{NP}$-complete was called the “Cook-Levin theorem,” and the proof of this result was not exactly trivial!

Strategy for Proving $\mathcal{NP}$-Completeness

Given a ***decision problem*** that you suspect to be $\mathcal{NP}$-complete...

1. If this has not already been done for you[1], describe an ***encoding scheme*** that can used to define a ***language*** L (over some reasonable alphabet) of encodings of Yes-instances: You will be proving that this *language* is $\mathcal{NP}$-complete.

[1] It *will* be done for you already on assignments and tests in this course.

Strategy for Proving $\mathcal{NP}$-Completeness

2. Prove that $L \in \mathcal{NP}$:
   (a) Describe ***certificates*** for Yes-instances.
   (b) Describe an ***encoding scheme*** for these certificates.
   (c) Describe a ***verification algorithm*** for L.
   (d) Confirm that the number of steps used by this algorithm is at at most polynomial in the input string — *not* including the length of a certificate — in the worst case. You will generally need to confirm that all Yes-instances have ***short*** certificates as part of this.

Strategy for Proving $\mathcal{NP}$-Completeness

- ***Note:*** You do not need to describe a deterministic Turing machine to complete step 2(d): It suffices to describe a ***deterministic*** algorithm (generally given as pseudocode) that could be implemented as a Java or Python program, and that uses a number of steps polynomial in the the length of the input string in the worst case.

Strategy for Proving $\mathcal{NP}$-Completeness

3. Prove that L is $\mathcal{NP}$-hard: Suppose $L \subseteq \Sigma^\star$.
   (a) Choose some language, $\widehat{L} \subseteq \widehat{\Sigma}^\star$ that is *already* known to be $\mathcal{NP}$-complete.
   (b) Describe a well-defined total function $f : \widehat{\Sigma}^\star \to \Sigma^\star$ such that
      - for every string $\omega \in \widehat{\Sigma}^\star$, $\omega \in \widehat{L} \iff f(\omega) \in L$, and
      - there is a deterministic Turing machine (which you should describe, as pseudocode — or by giving a deterministic Java or Python program) that computes f using at most a polynomial number of steps (in the length of its input string) in the worst case.

   Note that it follows that $\widehat{L} \preceq_{\text{P, M}} L$, so that L is $\mathcal{NP}$-hard since $\widehat{L}$ is.

Strategy for Proving $\mathcal{NP}$-Completeness

- ***Note:*** Once again, you do not need to describe a Turing machine that computes f — but your answer should be detailed enough so that it shows that there *is* a Turing machine that computes f, using a number of steps that is at most polynomial in the length of the input string.

4. Conclude that L is $\mathcal{NP}$-complete.

Strategy for Proving $\mathcal{NP}$-Completeness

One More Optional Step

- After defining encodings for instances, I often define a ***language of instances*** L_I such that $L \subseteq L_I \subseteq \Sigma^\star$, including all (valid) encodings of *instances* of the decision problem being considered — and prove that $L_I \in \mathcal{P}$.
- This is not needed for a proof that L is $\mathcal{NP}$-complete. However, it can make it easier to *use* the fact that L is $\mathcal{NP}$-complete, when proving that *other* languages are also $\mathcal{NP}$-complete, later on.

A Problem With These Examples

These notes continue with proofs that another three languages are $\mathcal{NP}$-complete.

- These proofs are ***too complicated*** to be thought of as good examples of the kinds of proofs that students might need to write in this course!
- They involve some of the *first* languages that were shown to $\mathcal{NP}$-complete after the Cook-Levin Theorem was first presented.
- At that point only a small number of $\mathcal{NP}$-complete problems were already known — so there only a few ways to *start* a proof of $\mathcal{NP}$-hardness, and these proofs were more complicated than is generally necessary, now, because of that.

CNF-Satisfiability: The Problem

Consider ***Boolean formulas*** over the set of Boolean variables $\mathcal{V} = \{x_0, x_1, x_2, \ldots\}$, as defined in Lecture #11.

Definition:

- A ***literal*** is either a variable x_i or its negation, $\neg x_i$, for $i \geq 0$
- A ***clause*** is the "or" of one more literals $\ell_1, \ell_2, \ldots, \ell_k$ (for some positive integer k):

$$(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k)$$

- A Boolean formula $\mathcal{F}$ in ***conjunctive normal form*** is the "and" of one or more clauses $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$ (for some positive integer n):

$$(\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \cdots \wedge \mathcal{C}_n)$$

CNF-Satisfiability: The Problem

The ***CNF Satisfiability Problem*** is the following decision problem.

CNF Satisfiability

Instance: A Boolean formula $\mathcal{F}$ in conjunctive normal form

Question: Is $\mathcal{F}$ satisfiable?

CNF-Satisfiability: Encodings

As in the previous lecture, let

$$\Sigma_F = \{\mathtt{x}, 0, 1, 2, \ldots, 9, \wedge, \vee, \neg, (,)\}$$

- Each ***Boolean variable*** x_i should be encoded by the string $e(x_i) \in \Sigma_F^\star$ consisting of the letter $\mathtt{x}$, followed by the unpadded decimal representation of i.
- The other symbols (brackets, $\wedge$, $\vee$, and $\neg$) in a Boolean formula $\mathcal{F}$ in conjunctive normal form can be encoded by themselves — so that (for example) if $\mathcal{F}$ is the formula

  $$((x_2 \vee x_{105} \vee \neg x_3) \wedge (x_{11}) \wedge (x_1 \vee x_4 \vee x_5 \vee x_6))$$

  then the encoding $e(\mathcal{F})$ is the string

  $$((\mathtt{x}2 \vee \mathtt{x}105 \vee \neg\mathtt{x}3) \wedge (\mathtt{x}11) \wedge (\mathtt{x}1 \vee \mathtt{x}4 \vee \mathtt{x}5 \vee \mathtt{x}6))$$

CNF-Satisfiability: Languages of Interest

Two languages — both subsets of $\Sigma_F^\star$ — can now be defined.

- Let $L_{\text{CNF}} \subseteq \Sigma_F^\star$ be the set of encodings of Boolean formulas $\mathcal{F}$ in conjunctive normal form.
- Let $L_{\text{CNF-SAT}} \subseteq L_{\text{CNF}}$ be the set of encodings of *satisfiable* Boolean formulas $\mathcal{F}$ in conjunctive normal form.

CNF-Satisfiability: $L_{\text{CNF}} \in \mathcal{P}$

Claim #1(a): $L_{\text{CNF}} \in \mathcal{P}$.

Sketch of Proof: Given a string $\omega \in \Sigma_F^\star$ it is possible to decide whether ω encodes a Boolean formula in conjunctive normal form (i.e., to decide whether $\omega \in L_{\text{CNF}}$) by sweeping over the input string

- checking whether copies of $\mathtt{x}$ seen are the beginnings of (well-formed) encodings of Boolean variables, and
- keeping track of — and matching — brackets seen, in order to determine which Boolean operator might next be expected to appear.

Indeed can be shown that L_{CNF} can be decided by a deterministic Turing machine, using a number of moves at most linear in the length of the input string — so $L_{\text{CNF}} \in \mathcal{P}$. $\square$

CNF-Satisfiability: $L_{\text{CNF-SAT}} \in \mathcal{NP}$

Claim #1(b): $L_{\text{CNF-SAT}} \in \mathcal{NP}$.

Sketch of Proof: A ***certificate*** for a string $\omega \in L_{\text{CNF-SAT}}$ — which encodes some satisfiable Boolean formula $\mathcal{F}$ in conjunctive normal form — is an encoding of a truth assignment $\varphi : \mathcal{V} \to \{\mathtt{T}, \mathtt{F}\}$ such that $\varphi(\mathcal{F}) = \mathtt{T}$.

- Truth assignments can be encoded using the alphabet Σ_C described in the previous lecture — and in exactly the same way as described in that lecture: A truth assignment φ is represented by encoding the finite subset of variables $\mathcal{S}$ of $\mathcal{V}$ that *appear in* $\mathcal{F}$ and that have truth value $\mathtt{T}$ under φ.
- It can be established (as in the previous lecture) that if $\omega \in L_{\text{CNF-SAT}}$ then there exists a certificate $\nu \in \Sigma_C^{\star}$ such that $|\nu| \leq |\omega| + 2$.

CNF-Satisfiability: $L_{\text{CNF-SAT}} \in \mathcal{NP}$

- Consider the ***verification algorithm*** for L_{FSAT} from the previous lecture. This begins by checking whether the input begins with a string $\omega\#$ such that $\omega \in L_{\text{F}}$ — ***rejecting*** if this is not the case.
- If this is replaced by initial step in which one checks whether the input begins with a string $\omega\#$ such that $\omega \in L_{\text{CNF}}$, instead (***rejecting*** if the test fails for this case as well) then it is straightforward to modify the analysis, from the previous lecture, to show that the result is a polynomial-time verification algorithm for $L_{\text{CNF-SAT}}$.
- Thus $L_{\text{CNF-SAT}} \in \mathcal{NP}$. □

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

Claim #1(c): $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-hard.

Sketch of Proof: It will be shown that

$$L_{\text{FSAT}} \preceq_{\text{P, M}} L_{\text{CNF-SAT}}.$$

Since L_{FSAT} is $\mathcal{NP}$-hard (as shown in the previous lecture), this implies that $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-hard, as claimed.

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

- ***Good News:*** Every Boolean formula $\mathcal{F}$ has a Boolean formula $\widetilde{\mathcal{F}}$, in conjunctive normal form, that is logically equivalent to it.
- ***Bad News:*** Sometimes *every* Boolean formula in conjunctive normal form that is logically equivalent to $\mathcal{F}$ also has length *exponential* in the length of $\mathcal{F}$.
- ***Good News:*** A reasonably short Boolean formula $\widehat{\mathcal{F}}$, *that depends on more variables than* $\mathcal{F}$, can be used to develop the reduction on the previous slide.

  In particular $\widehat{\mathcal{F}}$ will depend all the Boolean variables that $\mathcal{F}$ does, along with a Boolean variable[2] $y_{\mathcal{G}}$ for each ***subformula*** of $\mathcal{F}$.

---

[2]The new variables can be renamed, as the formula is generated, so that $\widehat{\mathcal{F}}$ only includes Boolean variables in $\mathcal{V}$.

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

Let us define a ***set*** $S_{\mathcal{G}}$ of ***clauses*** for every subformula $\mathcal{G}$ of $\mathcal{F}$:

- If $\mathcal{G}$ is a Boolean variable x_i then

$$S_{\mathcal{G}} = \{(y_{\mathcal{G}} \vee \neg x_i), (\neg y_{\mathcal{G}} \vee x_i)\}$$

  Note: The clauses in $S_{\mathcal{G}}$ are all satisfied if and only if $y_{\mathcal{G}}$ and x_i have the same truth value.

- If $\mathcal{G}$ is $\neg\widehat{\mathcal{G}}$ for another subformula $\widehat{\mathcal{G}}$ then

$$S_{\mathcal{G}} = \{(y_{\mathcal{G}} \vee y_{\widehat{\mathcal{G}}}), (\neg y_{\mathcal{G}} \vee \neg y_{\widehat{\mathcal{G}}})\} \cup S_{\widehat{\mathcal{G}}}.$$

  Note: The clauses in $\{(y_{\mathcal{G}} \vee y_{\widehat{\mathcal{G}}}), (\neg y_{\mathcal{G}} \vee \neg y_{\widehat{\mathcal{G}}})\}$ are both satisfied if and only if $y_{\mathcal{G}}$ and $\neg y_{\widehat{\mathcal{G}}}$ have the same truth value — which is what we want in *this* case.

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

- If $\mathcal{G} = (\mathcal{G}_1 \wedge \mathcal{G}_2 \wedge \cdots \wedge \mathcal{G}_k)$ for $k \geq 1$, then

$$S_{\mathcal{G}} = \{(y_{\mathcal{G}} \vee \neg y_{\mathcal{G}_1} \vee \neg y_{\mathcal{G}_2} \vee \ldots \neg y_{\mathcal{G}_k})\} \cup \{(\neg y_{\mathcal{G}} \vee y_{G_i}) \mid 1 \leq i \leq k\} \cup \bigcup_{1 \leq i \leq k} S_{G_i}$$

Note: The clauses in the initial subset

$$\{(y_{\mathcal{G}} \vee \neg y_{\mathcal{G}_1} \vee \neg y_{\mathcal{G}_2} \vee \ldots \neg y_{\mathcal{G}_k})\} \cup \{(\neg y_{\mathcal{G}} \vee y_{G_i}) \mid 1 \leq i \leq k\}$$

are all satisfied if and only if $y_{\mathcal{G}}$ and $(y_{\mathcal{G}_1} \wedge y_{\mathcal{G}_2} \wedge \cdots \wedge y_{\mathcal{G}_k})$ have the same truth value.

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

- If $\mathcal{G} = (\mathcal{G}_1 \vee \mathcal{G}_2 \vee \cdots \vee \mathcal{G}_k)$ for $k \geq 2$,[3] then

$$S_{\mathcal{G}} = \{(\neg y_{\mathcal{G}} \vee y_{\mathcal{G}_1} \vee y_{\mathcal{G}_2} \vee \cdots \vee y_{\mathcal{G}_k})\} \\ \{(y_{\mathcal{G}} \vee \neg y_{\mathcal{G}_i}) \mid 1 \leq i \leq k\} \cup \bigcup_{1 \leq i \leq k} S_{G_i}$$

Note: The clauses in the initial subset

$$\{(\neg y_{\mathcal{G}} \vee y_{\mathcal{G}_1} \vee y_{\mathcal{G}_2} \vee \cdots \vee y_{\mathcal{G}_k})\} \\ \{(y_{\mathcal{G}} \vee \neg y_{\mathcal{G}_i}) \mid 1 \leq i \leq k\}$$

are all satisfied if and only if $y_{\mathcal{G}}$ and $(y_{\mathcal{G}_1} \vee y_{\mathcal{G}_2} \vee \cdots \vee y_{\mathcal{G}_k})$ have the same truth value.

[3] If $k = 1$ then this formula is the same as the one on the previous slide for this case. The definitions of $S_{\mathcal{G}}$ would agree, if the case $k = 1$ was also included here, as well.

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

Now let $\widehat{\mathcal{F}}$ be the Boolean formula

$$\left((y_{\mathcal{F}}) \wedge \bigwedge_{C \in S_{\mathcal{F}}} C \right) .$$

- Note that $\widehat{\mathcal{F}}$ is a Boolean formula in conjunctive normal form.

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

Lemma: $\mathcal{F}$ is satisfiable if and only if $\widehat{\mathcal{F}}$ is satisfiable.

Proof:

Case: $\mathcal{F}$ is satisfiable. Then there is a satisfying truth assignment for this Boolean formula.

- This can be extended (providing truth values for variables $y_{\mathcal{G}}$ for every subformula $\mathcal{G}$) in such a way that
  - the truth value of $y_{\mathcal{G}}$ is the same as the truth value for $\mathcal{G}$, for every subformula $\mathcal{G}$ of $\mathcal{F}$, and
  - all the clauses in $S_{\mathcal{F}}$ are satisfied.
- Since $\mathcal{F}$ is satisfied under the original truth assignment $y_{\mathcal{F}}$ receives the truth value "true," so the clause $(y_{\mathcal{F}})$ is satisfied too.
- One can see by inspection of $\widehat{\mathcal{F}}$ that $\widehat{\mathcal{F}}$ is satisfied too, as required to establish that $\widehat{\mathcal{F}}$ is satisfiable.

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

Case: $\mathcal{F}$ is unsatisfiable. Consider any truth assignment φ for the variables in $\mathcal{F}$.

- The ***only*** way to extend this, so that all clauses in $S_{\mathcal{F}}$ are satisfied, is to set $y_{\mathcal{G}}$ to have the same truth value as $\mathcal{G}$ has under φ, for every subformula $\mathcal{G}$ of $\mathcal{F}$.
- If truth values for the new variables are ***not*** set in this way then $\widehat{\mathcal{F}}$ is not satisfied, because $\widehat{\mathcal{F}}$ includes all the clauses in $S_{\mathcal{F}}$.
- If truth values for the new variable ***are*** set in this way, then $\widehat{\mathcal{F}}$ is not satisfied, because $y_{\mathcal{F}}$ receives the truth value “F”, and $(y_{\mathcal{F}})$ is a clause in $\widehat{\mathcal{F}}$.
- Since we started with an arbitrarily chosen truth assignment, it follows that $\widehat{\mathcal{F}}$ is unsatisfiable — as required to establish the claim. □

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

- Now consider a function $f : \Sigma_F^\star \to \Sigma^\star$ such that, for $\omega \in \Sigma_F^\star$,
  - If $\omega \in L_{\text{F}}$ and, in particular, $\omega = \text{e}(\mathcal{F})$ for a Boolean formula $\mathcal{F}$, then $f(\omega) = \text{e}(\widehat{\mathcal{F}})$ for the corresponding Boolean formula $\widehat{\mathcal{F}}$, in conjunctive normal form, described on previous slides,[4] and
  - if $\omega \notin L_{\text{F}}$ then $f(\omega) = \lambda$, the empty string — so that $f(\omega) \notin L_{\text{CNF}}$.
- It follows by the above information that $\omega \in L_{\text{FSAT}}$ if and only if $f(\omega) \in L_{\text{CNF-SAT}}$ for every string $\omega \in \Sigma_F^\star$.

[4] …with variables renamed, so that this a formula with variables in $\mathcal{V}$, as previously noted…

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

Lemma: The above function f is computable using a number of steps that is at most polynomial in the length of the input string.

Idea of Proof:

- The algorithm from the previous lecture, to decide membership of a string in L_{F}, can modified to compute f.
- Instead of replacing a string (representing a subformula) with "F" one can replace this with the new variable whose truth value should match that of the subformula being processed. Corresponding clauses, that will be included in $\mathcal{F}$, should be added to a set of these, which is being maintained, at the same time.

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Hard

- If it is determined, during that the input string is not in L_{F}, then the computation should halt with the empty string on the output tape. If is determined, instead, that the input belongs to L_{F}, then the set of clauses now assembled can be used to generate an encoding of e($\mathcal{F}$).
- The process is simple enough to be implemented using a multi-tape deterministic using a number of moves that is at most cubic in the length of the input string, as needed to establish the claim. □

It follows, by the last two lemmas, that $L_{\text{FSAT}} \preceq_{\text{P, M}} L_{\text{CNF-SAT}}$. Since L_{FSAT} is $\mathcal{NP}$-hard this implies that $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-hard too.

CNF-Satisfiability: $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-Complete

- It has now been shown that $L_{\text{CNF-SAT}} \in \mathcal{NP}$ and that $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-hard.
- It follows that $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-complete.

3-CNF Satisfiability: The Problem

Definition: A Boolean formula $\mathcal{F}$ in conjunctive normal form is in ***3-conjunctive normal form*** — and also called a ***3-CNF formula*** — if every clause in $\mathcal{F}$ includes exactly literals, so $\mathcal{F}$ has the form

$$((\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge (\ell_{2,1} \vee \ell_{2,2} \vee \ell_{2,3}) \wedge \dots \\ \wedge (\ell_{k,1} \vee \ell_{k,2} \vee \ell_{k3}))$$

for some positive integer k and where $\ell_{i,j}$ is a literal — either x_h or $\neg x_h$ for $h \in \mathbb{N}$.

3-CNF Satisfiability: The Problem

The ***3-CNF Satisfiability Problem*** is the following decision problem.

CNF Satisfiability

Instance:	A Boolean formula $\mathcal{F}$ in 3-conjunctive normal form
Question:	Is $\mathcal{F}$ satisfiable?

3-CNF Satisfiability: Encodings

- Instances of this problem are also instances of the *CNF Satisfiability Problem,* and they can be encoded in exactly the same way.

3-CNF Satisfiability: Languages of Interest

Two languages — both subsets of $\Sigma_F^\star$ — can now be defined.

- Let $L_{\text{3CNF}} \subseteq \Sigma_F^\star$ be the set of encodings of Boolean formulas $\mathcal{F}$ in 3-conjunctive normal form.
- Let $L_{\text{3CNF-SAT}} \subseteq L_{\text{3CNF}}$ be the set of encodings of *satisfiable* Boolean formulas $\mathcal{F}$ in 3-conjunctive normal form.

3-CNF Satisfiability: $L_{\text{3CNF}} \in \mathcal{P}$

Claim #2(a): $L_{\text{3CNF}} \in \mathcal{P}$.

How To Prove This:

- It suffices to modify the deterministic algorithm to decide the language L_{CNF}, already described, by making one change. Recall that this makes a linear sweep over the input.
- The only change is to add a test that each clause, found during the sweep , includes exactly three literals. The input should be ***accepted*** if the original algorithm would accept it, and this additional test would also be passed. The output should be ***rejected*** otherwise.
- The correctness and efficiency of the modified process are easily proved.

3-CNF Satisfiability: $L_{\text{3CNF-SAT}} \in \mathcal{NP}$

Claim #2(b): $L_{\text{3CNF-SAT}} \in \mathcal{NP}$.

How To Prove This:

- The polynomial-time verification algorithm for $L_{\text{CNF-SAT}}$, already described, is easily modified to produce a polynomial-time verification algorithm for $L_{\text{3CNF-SAT}}$.
- The only difference is that a test whether $\omega \in L_{\text{CNF}}$, in the original algorithm, should be replaced with a test whether $\omega \in L_{\text{3CNF}}$ in the new one.
- Correctness can be established by an examination of the original algorithm and a review of the proof of *its* correctness. Efficiency can be established using Claim #2(a).

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Claim #2(c): $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-hard.

Sketch of Proof: It will be shown that

$$L_{\text{CNF-SAT}} \preceq_{\text{P, M}} L_{\text{3CNF-SAT}}.$$

Since $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-hard (by Claim #1(c)), this implies that $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-hard, as claimed.

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

- Consider a total function $f : \Sigma_F^\star \to \Sigma_F^\star$ with the following properties:
  - If $\omega \in \Sigma_F^\star$ but ω does not encode a Boolean formula in conjunctive normal form — so that $\omega \notin L_{\text{CNF-SAT}}$ — then $f(\omega) = \lambda$ (the empty string), so that $f(\omega) \notin L_{\text{3CNF-SAT}}$.
  - If $\omega \in \Sigma_F^\star$ *does* encode a Boolean formula $\mathcal{F}$ in conjunctive normal form, then $f(\omega)$ encodes a Boolean formula $\widehat{\mathcal{F}}$ in 3-conjunctive normal form such that $\mathcal{F}$ is satisfiable if and only if $\widehat{\mathcal{F}}$ is satisfiable.

  Then $\omega \in L_{\text{CNF-SAT}}$ if and only if $f(\omega) \in L_{\text{3CNF-SAT}}$ for all $\omega \in \Sigma_F^\star$

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

- ***Good News:*** Once again a reasonably short Boolean formula $\widehat{\mathcal{F}}$, *that depends on more variables than* $\mathcal{F}$, can be used to define the function f described in the previous slide.
- ***Even Better News:*** The reduction to be described next is *much* simpler than the previous one.
- Once again, the first version of $\widehat{\mathcal{F}}$ will include Boolean variables with names different from x_i for $i \in \mathbb{N}$. The same kind of “preprocessing” step, and use of global variable `next`, allows the new variables to be renamed, on the fly, so that this is not the case.

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Recall that $\mathcal{F}$ has the form

$$(\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \cdots \wedge \mathcal{C}_n)$$

where $\mathcal{C}_i$ is a clause for $1 \leq i \leq n$.

Strategy: We will do the following.

(a) Use $\mathcal{C}_i$ to define a set $\mathcal{S}_i$ of clauses, each including three literals, for $1 \leq i \leq n$.

(b) Set

$$\widehat{\mathcal{F}} = \left(\bigwedge_{1 \leq i \leq n} \; \bigwedge_{c \in \mathcal{S}_i} c \right)$$

— noting that $\widehat{\mathcal{F}}$ is a Boolean formula in 3-conjunctive normal form.

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

(c) Prove that $\mathcal{F}$ is satisfiable if and only if $\widehat{\mathcal{F}}$ is.

(d) Note that the structure of $\widehat{\mathcal{F}}$ and process used to define it is so simple that the function f can be computed using either a deterministic Java or Python program, or a deterministic Turing machine, using a number of steps that is at most polynomial in the length of the input string in the worst case.

It will then follow that $L_{\text{CNF-SAT}} \preceq_{\text{P, M}} L_{\text{3CNF-SAT}}$, implying that $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-hard.

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Defining $\mathcal{S}_i$ When $\mathcal{C}_i$ Only Includes One Literal:

If $\mathcal{C}_i$ is a clause (ℓ), where ℓ is a literal, then

$$\mathcal{S}_i = \{(\ell \vee \ell \vee \ell)\}$$

- A truth assignment satisfies $\mathcal{C}_i$ if and only if it satisfies all the clauses in $\mathcal{S}_i$.

  This will be true for next two cases too.

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Defining $\mathcal{S}_i$ When $\mathcal{C}_i$ Includes Exactly Two Literals:

If $\mathcal{C}_i$ is a clause $(\ell_1 \vee \ell_2)$, for literals ℓ_1 and ℓ_2, then

$$\mathcal{S}_i = \{(\ell_1 \vee \ell_2 \vee \ell_2)\}$$

Defining $\mathcal{S}_i$ When $\mathcal{C}_i$ Includes Exactly Three Literals:

If $\mathcal{C}_i$ is a clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ for literals ℓ_1, ℓ_2 and ℓ_3, then

$$\mathcal{S}_i = \{(\ell_1 \vee \ell_2 \vee \ell_3)\} = \{\mathcal{C}_i\}$$

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Defining $\mathcal{S}_i$ When $\mathcal{C}_i$ Includes Four or More Literals:

If $\mathcal{C}_i$ is a clause $(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k)$ for literals $\ell_1, \ell_2, \ldots, \ell_k$ where $k \geq 4$, introduce ***new*** variables $z_{i,1}, z_{i,2}, \ldots, z_{i,k-3}$ — which will ***only*** appear in the clauses included in $\mathcal{S}_i$.

$$\mathcal{S}_i = \{(\ell_1 \vee \ell_2 \vee z_{i,1})\} \cup \bigcup_{1 \leq j \leq k-4} \{(\neg z_{i,j} \vee \ell_{j+2} \vee z_{i,j+1})\} \cup \{(\neg z_{i,k-3} \vee \ell_{k-1} \vee \ell_k)\}$$

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Examples:

- If $k = 4$ then $\mathcal{S}_i = \{(\ell_1 \vee \ell_2 \vee z_{i,1}), (\neg z_{i,1} \vee \ell_3 \vee \ell_4)\}$
- If $k = 5$ then
  $$\mathcal{S}_i = \{(\ell_1 \vee \ell_2 \vee z_{i,1}), (\neg z_{i,1} \vee \ell_3 \vee z_{i,2}), (\neg z_{i,2} \vee \ell_4 \vee \ell_5)\}$$
- If $k = 6$ then
  $$\mathcal{S}_i = \{(\ell_1 \vee \ell_2 \vee z_{i,1}), (\neg z_{i,1} \vee \ell_3 \vee z_{i,2}), \\ (\neg z_{i,2} \vee \ell_4 \vee z_{i,3}), (\neg z_{i,3} \vee \ell_5 \vee \ell_6)\}$$

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Lemma: Every truth assignment that satisfies $\mathcal{C}_i$ can be extended — by assigning truth values to $z_{i,1}, z_{i,2}, \ldots, z_{i,k-3}$ — to produce a truth assignment that satisfies all the clauses in $\mathcal{S}_i$.

Proof: Consider three cases — one of which must arise.

Case: One of ℓ_1 or ℓ_2 is satisfied under the truth assignment.

- The first clause included in $\mathcal{S}_i$, $(\ell_1 \vee \ell_2 \vee z_{i,1})$, is satisfied.
- Setting the truth values of all or $z_{i,1}, z_{i,2}, \ldots, z_{i,k-3}$ to be $\mathtt{F}$ ensures that all the other clauses in $\mathcal{S}_i$ are satisfied too, because each includes a literal $\neg z_{i,j}$ where $1 \leq j \leq k-3$.

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Case: The first of $\ell_1, \ell_2, \ldots, \ell_k$ satisfied is ℓ_j, where $3 \leq j \leq k-2$.

- The only clause in $\mathcal{S}_i$ including ℓ_j is $(\neg z_{i,j-2} \vee \ell_j \vee z_{j-1})$, and this clause is satisfied.
- Setting the truth of $z_{i,1}, z_{i,2}, \ldots, z_{i,j-2}$ to be $\mathtt{T}$ and the truth value of $z_{i,j-1}, z_{i,j}, \ldots, z_{i,k-2}$ to be $\mathtt{F}$ ensures that all the other clauses in $\mathcal{S}_i$ are satisfied too: The ones before this one in the natural ordering for $\mathcal{S}_i$ each include a literal $z_{i,h}$ where $1 \leq h \leq j-2$, and the ones after it includes a literal $\neg z_{i,h}$ where $j-1 \leq h \leq k-3$.

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Case: The first of $\ell_1, \ell_2, \dots, \ell_k$ satisfied is either ℓ_{k-1} or ℓ_k.

- The last clause in $\mathcal{S}_i$, $(\neg z_{i,k-3} \vee \ell_{k-1} \vee \ell_k)$, is satisfied.
- Setting the true value for $z_{i,1}, z_{i,2}, \dots, z_{i,k-3}$ to be $\mathtt{T}$ ensures that all the other clauses in $\mathcal{S}_i$ are satisfied too, because each includes one of $z_{i,1}, z_{i,2}, \dots, z_{i,k-3}$ as a literal.

Since the desired result has been established in all possible cases, this establishes the claim. $\square$

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

Lemma: It is impossible to extend a truth assignment that *does not* satisfy $\mathcal{C}_i$, by setting truth values for $z_{i,1}, z_{i,2}, \dots, z_{i,k-3}$, in order to satisfy ***all*** of the clauses in $\mathcal{S}_i$.

Proof:

- It is necessary to set $z_{i,1}$ to be $\mathtt{T}$ to satisfy the first clause, $(\ell_1 \vee \ell_2 \vee z_{i,1})$.
- For $j = 2, 3, \dots, z_{k-3}$ it is now necessary to set $z_{i,j}$ to be $\mathtt{T}$ the satisfy a later clause, $(\neg z_{i,j-1} \vee \ell_{i,j+1} \vee z_{i,j})$.
- However, truth values for all variables have now been set and the final clause, $(\neg z_{i,k-3} \vee \ell_{k-1} \vee \ell_k)$, is not satisfied. $\square$

Corollary: $\mathcal{F}$ is satisfiable if and only if $\widehat{\mathcal{F}}$ is satisfiable. $\square$

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Hard

- It has already been claimed that $L_{\text{3CNF}} \in \mathcal{P}$, so that one can check whether an input string $\omega \in \Sigma_F^{\star}$ is in L_{3CNF}, setting $f(\omega)$ to be λ if it is not, using a deterministic Turing machine in polynomial time.
- An encoding of $\widehat{\mathcal{F}}$ using extra variables $z_{i,j}$ can be generated deterministically in polynomial time — essentially, using a single sweep over the encoding of $\mathcal{F}$.
- Renaming of variables to complete the process can be carried out by finding the largest integer i such that x_i appears in $\mathcal{F}$, and then replacing new variables with $x_{i+1}, x_{i+2}, \dots$ — also using at most a polynomial number of steps in the length of the input..
- Thus f can be computed deterministically in polynomial time, so that $L_{\text{CNF-SAT}} \preceq_{\text{P, M}} L_{\text{3CNF-SAT}}$. Since $L_{\text{CNF-SAT}}$ is $\mathcal{NP}$-hard, it follows that $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-hard too.

3-CNF Satisfiability: $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-Complete

- It has now been shown that $L_{\text{3CNF-SAT}} \in \mathcal{NP}$ and that $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-hard.
- It follows that $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-complete.

k-Clique: The Problem

Suppose $G = (V, E)$ is an undirected graph.

Definition: A ***clique*** in G is a subset C of V such that $(u, v) \in E$ for all vertices u and v such that $u, v \in C$ and $u \neq v$.

Consider the following decision problem:

k-Clique

Instance: An undirected graph $G = (V, E)$ and a positive integer k

Question: Does G have a clique with size (at least) k?

k-Clique: Encodings

Consider an alphabet

$$\Sigma_G = \{\mathtt{v}, \mathtt{0}, \mathtt{1}, \mathtt{2}, \dots, \mathtt{9}, \mathtt{,}, \mathtt{(}, \mathtt{)}, \mathtt{\{}, \mathtt{\}}\}$$

This will be used to encode instances of this decision problem.

- Renaming vertices if needed, suppose (or require) that $V = \{v_0, v_1, \dots, v_{n-1}\}$ for some positive integer n.
- For $1 \le i \le n$, each vertex v_i has an encoding $e(v_i) \in \Sigma_G^\star$: $e(v_i)$ is the letter $\mathtt{v}$ followed by the unpadded decimal representation of the index i — $\mathtt{v0}$ if $i = 0$.
- ***Note:*** Suppose $V \neq \emptyset$, because this annoying special case is not important.

  Then $n \ge 1$ and each of the above encodings is a nonempty string with length at most $\lceil \log_{10} n \rceil + 1$.

k-Clique: Encodings

- Since G is an ***undirected*** graph every *edge* can be written as (v_i, v_j) where $0 \leq i < j \leq n-1$. The encoding $e((v_i, v_j))$ of this vertex begins with a left bracket, "(", continues with $e(v_i)$, a comma, ",", $e(v_j)$, and ends with a right bracket, ")".
- ***Note:*** This is a nonempty string with length at most $2\lceil \log_{10} n \rceil + 5$.

k-Clique: Encodings

- The encoding $e(E)$ of the set of edges E
  - starts with a left bracket,
  - continues with the encodings of each edge, separated by commas, in nondecreasing order of first vertex and, when the first vertices are the same, increasing order of second vertex, and
  - ends with a right bracket.

Note: Since there are at most $\binom{n}{2}$ edges this is a string with length at most $n^2\lceil \log_{10} n\rceil + 3n^2$.

k-Clique: Encodings

The encoding $e(G)$ of an undirected graph $G = (V, E)$, consists of

- A left bracket, "`(`",
- The number n of vertices in V — encoded in ***unary*** (as a string of n `1`'s),
- A comma, `,`
- The encoding $e(E)$ of the set of edges, as described above, and
- A right bracket, "`)`"

k-Clique: Encodings

The encoding of an instance of the *k*-Clique problem consists of

- A left bracket, "(",
- The encoding $e(G)$ of the input graph $G = (V, E)$, as described above,
- A comma, ")",
- The unpadded decimal representation of the input integer k, and
- A right bracket, ")"

k-Clique: Languages of Interest

Two languages — both subsets of $\Sigma_G^\star$ — can now be defined.

- Let $L_{\text{Graph+Bound}} \subseteq \Sigma_G^\star$ be the set of encodings of instances of the *k*-Clique problem — that is, encodings of undirected graphs and positive integers, as described above.
- Let $L_{k\text{-Clique}} \subseteq L_{\text{Graph+Bound}}$ be the set of Yes-instances of the "*k*-Clique" problem — that is, the set of encodings of undirected graphs $G = (V, E)$ and positive integers k such that G has a clique with size at least k.

k-Clique: $L_{\text{Graph+Bound}} \in \mathcal{P}$

Exercise:

(a) Use the description of encodings of undirected graphs, given above, to describe whether a string $\mu \in \Sigma_G^\star$ is an encoding of an undirected graph, deterministically, using a number of steps that is at most polynomial in the length of the input string μ.

(b) Use this to complete a proof that $L_{\text{Graph+Bound}} \in \mathcal{P}$.

By doing so, you will have proved

Claim #3(a): $L_{\text{Graph+Bound}} \in \mathcal{P}$.

k-Clique: $L_{k\text{-Clique}} \in \mathcal{NP}$

- A string $\mu \in \Sigma_G^\star$ such that $\mu \notin L_{\text{Graph+Bound}}$ is certainly not in $L_{k\text{-Clique}}$.
- A string $\mu \in L_{k\text{-Clique}}$ encoding an undirected graph $G = (V, E)$ and a positive integer k such that $k > |V|$ is also certainly not in $L_{k\text{-Clique}}$, either.
- It therefore suffices to consider encodings of undirected graphs $G = (V, E)$ and positive integers k such that $k \leq |V|$.

k-Clique: $L_{k\text{-Clique}} \in \mathcal{NP}$

- An encoding of a ***clique*** in G with size at least k will be used as a ***certificate*** for a string $\omega \in L_{\text{3-CNFSAT}}$ that encodes a graph $G = (V, E)$ and positive integer k.
- Since Σ_G includes all the symbols needed to encode sets of vertices in G we can set Σ_C to be Σ_G.
- A clique can then be encoded as a subset of vertices in G — sorted by increasing index, to make it easier to confirm that a subset of V really *is* being encoded.

k-Clique: $L_{k\text{-Clique}} \in \mathcal{NP}$

Exercise: Use the above information to complete a proof that $L_{k\text{-Clique}} \in \mathcal{NP}$ by describing a verification algorithm for this language, and proving that it solves the problem that is supposed to, using a number of moves that is bounded as required.

By doing so, you will have proved

Claim #3(b): $L_{k\text{-Clique}} \in \mathcal{NP}$.

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard

Claim #3(c): $L_{k\text{-Clique}}$ is $\mathcal{NP}$-hard.

Sketch of Proof: It will be shown that

$$L_{\text{3CNF-SAT}} \preceq_{\text{P, M}} L_{k\text{-Clique}}.$$

Since $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-hard (by Claim #2(c)), this implies that $L_{k\text{-Clique}}$ is $\mathcal{NP}$-hard, as claimed.

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard

Consider a Boolean formula $\mathcal{F}$ in 3-conjunctive normal form — so that $\mathcal{F}$ has the form

$$((\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1}, 3) \wedge (\ell_{2,1} \vee \ell_{2,2} \vee \ell_{2,3}) \wedge \dots \\ \wedge (\ell_{m,1} \vee \ell_{m,2} \vee \ell_{m,3}))$$

for some positive integer m, and where $\ell_{i,j}$ is a literal for $1 \leq i \leq m$ and $1 \leq j \leq 3$.

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard

- Consider an undirected graph $G = (V, E)$ with $3m$ vertices: For $1 \leq i \leq m$,
  - Vertex v_{3i-3} ***corresponds to*** the literal $\ell_{i,1}$,
  - vertex v_{3i-2} ***corresponds to*** the literal $\ell_{i,2}$, and
  - vertex v_{3i-1} ***corresponds to*** the literal $\ell_{i,3}$.

  Then every vertex in V corresponds to exactly one literal in $\mathcal{F}$.

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard

- For $0 \le s, t \le 3m - 1$ include (v_s, v_t) in E if and only if ***both*** of the following properties are satisfied:
  (a) v_s and v_t correspond to *different* clauses in $\mathcal{F}$, so $\lfloor s/3 \rfloor \ne \lfloor t/3 \rfloor$, and
  (b) the literals ℓ and $\widehat{\ell}$ corresponding to v_s and v_t are not ***inconsistent*** — that is, it is not true that one of them is x_h and the other is $\neg x_h$, for any $h \in \mathbb{N}$.

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard

Example: Suppose $\mathcal{F}$ is the 3-CNF Boolean formula

$$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \\ \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4))$$

Then

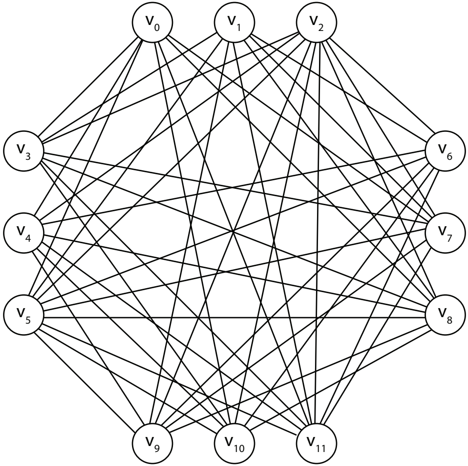
- v_0 corresponds to the literal $\ell_{11} = x_1$
- v_1 corresponds to the literal $\ell_{1,2} = x_2$
- v_2 corresponds to the literal $\ell_{1,3} = x_3$
- v_3 corresponds to the literal $\ell_{2,1} = x_1$
- v_4 corresponds to the literal $\ell_{2,2} = \neg x_2$
- v_5 corresponds to the literal $\ell_{2,3} = x_3$

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard

- v_6 corresponds to the literal $\ell_{3,1} = \neg x_1$
- v_7 corresponds to the literal $\ell_{3,2} = x_2$
- v_8 corresponds to the literal $\ell_{3,3} = x_4$
- v_9 corresponds to the literal $\ell_{4,1} = \neg x_1$
- v_{10} corresponds to the literal $\ell_{4,2} = \neg x_2$
- v_{11} corresponds to the literal $\ell_{4,3} = \neg x_4$

and G is as shown on the following slide; $k = 4$.

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard



k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard

Lemma: If $\mathcal{F}$ is satisfiable then G has a clique with size k.

Proof: Since $\mathcal{F}$ is satisfiable, it has a satisfying truth assignment — so it is possible to pick a literal $\ell_{i,j}$ (for $1 \leq j \leq 3$) that is satisfied under this truth assignment, for each integer i such that $1 \leq i \leq m = k$.

It follows by the definition of the set of edges included in E, above, that the set of vertices corresponding to these literals forms a clique with size k, as required. □

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard

Lemma: If G has a clique with size k then $\mathcal{F}$ is satisfiable.

Proof: Consider a clique C of G with size k.

- Since $k = m$ (the number of clauses in $\mathcal{F}$) property (a) in the rule for inclusion of edges in E ensures that C includes a vertex corresponding a literal to each one of the clauses in $\mathcal{F}$.
- Property (b) ensures that x_h and $\neg x_h$ are not both in the set of literals corresponding to vertices in C for any natural number h.
- It is therefore possible to define a truth assignment that satisfies all these literals — and (regardless of truth assignments for any other Boolean variables) that ensures that $\mathcal{F}$ is satisfied under this truth assignment — so that $\mathcal{F}$ is satisfiable, as claimed. $\square$

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Hard

Corollary: G has a clique with size (at least) k if and only if $\mathcal{F}$ is satisfiable.

Exercise:

- Supplying additional technical details. and describing and analyzing any algorithm (or Turing machine) that is required, use this information to complete a proof that

$$L_{\text{3CNF-SAT}} \preceq_{\text{P, M}} L_{k\text{-Clique}}.$$

- Since $L_{\text{3CNF-SAT}}$ is $\mathcal{NP}$-hard, it follows from this that $L_{k\text{-Clique}}$ is $\mathcal{NP}$-hard too.

k-Clique: $L_{k\text{-Clique}}$ is $\mathcal{NP}$-Complete

- It has now been argued that $L_{k\text{-Clique}} \in \mathcal{NP}$ and that $L_{k\text{-Clique}}$ is $\mathcal{NP}$-hard.
- It follows that $L_{k\text{-Clique}}$ is $\mathcal{NP}$-complete.

Other $\mathcal{NP}$-Complete Problems

- Chapter 34 of the third edition of *Introduction to Algorithms* includes an introduction to several other "classical" $\mathcal{NP}$-complete problems and sketches of proofs that they are $\mathcal{NP}$-complete. This is available as an ebook from the University of Calgary library.
- *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness* is an excellent older reference that includes information about how one can prove that a language is $\mathcal{NP}$-complete and that describes ***many*** more $\mathcal{NP}$-complete problems. This is available at the University of Calgary library.

Mistakes To Watch for and Avoid

- Students *will* be asked to prove that languages are $\mathcal{NP}$-complete on assignments and tests. The proofs that are required will be *much* simpler and shorter than the first proof in these notes! Indeed, they might be simpler than the other proofs in these notes too.
- Common mistakes you should watch for and avoid include
  - giving a reduction in the wrong direction
  - failing to ensure that there is a certificate ***with polynomial length*** when proving membership in $\mathcal{NP}$
  - failing to ensure that the function $f : \Sigma_1^\star \to \Sigma_2^\star$ (used to define a reduction) is
    - a well-defined ***total*** function, and
    - computable by a deterministic algorithm using a number of steps that is at most polynomial in the length of the input string in the worst case.