Non-Deterministic Polynomial Time

1 Non-Deterministic Polynomial Time NP

Definition (Non-deterministic polynomial time, complexity class NP)**.** Fix some alphabet Σ. We say that a language L can be decided in *non-deterministic polynomial time* if there exists

1. a polynomial-time decider TM V that takes two strings as input, and
2. a constant $k > 0$,

such that for all $x \in \Sigma^*$:

- if $x \in L$, then there exists $u \in \Sigma^*$ with $|u| \leq |x|^k$ such that $V(x, u)$ accepts,
- if $x \notin L$, then for all $u \in \Sigma^*$, $V(x, u)$ rejects.

If $x \in L$, a string u that makes $V(x, u)$ accept is called a *proof* (or *certificate*) of x being in L. The TM V is called a *verifier*.

We denote by NP the set of all languages which can be decided in non-deterministic polynomial time.

Proposition (3COL is in NP)**.** *3COL* $\in$ NP.

Proof. To show 3COL is in NP, we need to show that there is a polynomial-time verifier TM A with the properties stated in Definition (Non-deterministic polynomial time, complexity class NP) (we are using A to denote the verifier and not V because we will use V

to denote the vertex set of a graph). Recall that an instance of the 3COL problem is an undirected graph G. The description of A is as follows.

def $A(\langle \text{graph } G = (V, E)\rangle,\ u)$:

1. `If` u `does not correspond to a 3-coloring of` V`, reject.`
2. `If there is` $\{v, w\} \in E$ `where` v `and` w `have the same color, reject.`
3. `Else, accept.`

We now show that A satisfies the two conditions stated in Definition (Non-deterministic polynomial time, complexity class NP). If x is in the language, that means x is a valid encoding of a graph $G = (V, E)$ and this graph is 3-colorable. When u is a 3-coloring of V, $|u| = O(|V|)$. And for this x and u, the verifier accepts. On the other hand, if x is not in the language, then either (i) x is not a valid encoding of a graph or (ii) it is a valid encoding of a graph which is not 3-colorable. In case (i), the verifier rejects (which is done implicitly since the input is not of the correct type). In case (ii), any u that does not correspond to a 3-coloring of the vertices makes the verifier reject. Furthermore, any u that does correspond to a 3-coloring of the vertices must be such that there is an edge whose endpoints are colored with the same color. Therefore, in this case, the verifier again rejects, as desired.

Now we show that the machine is polynomial-time. To check whether u is a 3-coloring of the vertices takes polynomial time since you just need to check that you are given $|V|$ colors, each being one of 3 colors. To check that it is indeed a legal 3-coloring is polynomial time as well since you just need to go through every edge once and check for each that the endpoints are different colors.

This completes the proof that 3COL is in NP. □

Note (Steps to show a languages is in NP)**.** Showing that a language L is in NP involves the following steps:

1. Present a TM V (that takes two inputs x and u).
2. Argue that V has polynomial running time.
3. Argue that V works correctly, which involves arguing the following for some constant $k > 0$:
   - for all $x \in L$, there exists $u \in \Sigma^*$ with $|u| \leq |x|^k$ such that $V(x, u)$ accepts;
   - for all $x \notin L$ and for all $u \in \Sigma^*$, $V(x, u)$ rejects.

Proposition (CIRCUIT-SAT is in NP)**.** $\text{CIRCUIT} - \text{SAT} \in \text{NP}$.

Proof. To show CIRCUIT-SAT is in NP, we need to show that there is a polynomial-time verifier V with the properties stated in Definition (Non-deterministic polynomial time, complexity class NP). We start by presenting V.

def $V(\langle \text{circuit } C\rangle,\ u)$:

1. `If` u `does not correspond to a` $0/1$ `assignment to input gates, reject.`
2. `Compute the output of the circuit` $C(u)$.
3. `If the output is` 0`, reject.`
4. `Else, accept.`

We first show that the verifier V satisfies the two conditions stated in Definition (Non-deterministic polynomial time, complexity class NP). If x is in the language, that means that x corresponds to a valid encoding of a circuit and there is some $0/1$-assignment to the input gates that makes the circuit output 1. When u is such a $0/1$-assignment, then $|u| = O(n)$ (where n is the length of x), and the verifier accepts the input (x, u). On the other hand, if x is not in the language, then either (i) x is not a valid encoding of a circuit or (ii) it is a valid encoding of a circuit which is not satisfiable. In case (i), the verifier rejects (which is done implicitly since the input is not of the correct type). In case (ii), any u that does not correspond to a $0/1$-assignment to the input gates makes the verifier reject. Furthermore, any u that does correspond to a $0/1$-assignment to the input gates must be such that, with this assignment, the circuit evaluates to 0. Therefore, in this case, the verifier again rejects, as desired.

Now we show the verifier is polynomial-time. To check whether u is a valid $0/1$-assignment to the input gates takes polynomial time since you just need to check that you are given t bits, where t is the number of input gates. The output of the circuit can be computed in polynomial time since it takes constant number of steps to compute each gate.

This completes the proof of CIRCUIT-SAT is in NP. □

Exercise (CLIQUE is in NP)**.** Show that CLIQUE $\in$ NP.

Solution. To show CLIQUE is in NP, we need to show that there is a polynomial-time verifier A with the properties stated in Definition (Non-deterministic polynomial time, complexity class NP). We start by presenting A. (We are using A to denote the verifier and not V because we will use V to denote the vertex set of a graph.)

def $A(\langle$graph $G = (V, E)$, positive natural $k\rangle,\ u)$:

1. `If` u `does not correspond to a set` $S \subseteq V$ `with` $|S| = k$`, reject.`
2. `For each pair of vertices` u, v `in` S`:`
3. `  If` $\{u, v\} \notin E$`, reject.`
4. `Accept.`

We first show that the verifier A satisfies the two conditions stated in Definition (Non-deterministic polynomial time, complexity class NP). If x is in the language, that means that x corresponds to a valid encoding of a graph $G = (V, E)$ together with a number $k \in \mathbb{N}$. Furthermore, it must be the case that there is some $S \subseteq V$ with $|S| = k$ such that S forms a k-clique. When u is the encoding of such an S, then $|u| = O(n)$ where n is the length of x, and the verifier A accepts (x, u). On the other hand, if x is not in the language, then either (i) x is not a valid encoding of a graph $G = (V, E)$ together with a number $k \in \mathbb{N}$ or (ii) it is a valid encoding $\langle G, k\rangle$, but there is no $S \subseteq V$, $|S| = k$, that forms a k-clique. In case (i), the verifier rejects (which is done implicitly since the input is not of the correct type). In case (ii), any u that does not correspond to a set $S \subseteq V$ with $|S| = k$ makes the verifier reject. Furthermore, any u that does correspond to such an S cannot form a k-clique. In this case, the for loop will detect this and the verifier again rejects, as desired.

Now we show the verifier is polynomial-time. Checking whether u is a valid encoding of a set $S \subseteq V$ with $|S| = k$ is polynomial time. If this check passes, then the for-loop repeats at most $O(|V|^2)$ many times, and the body of the loop can be carried out in polynomial time. So in total, the work being done is polynomial time. ■

Exercise (IS is in NP)**.** Show that IS $\in$ NP.

Solution. The argument is very similar to the one above. Here is the verifier:

def $A(\langle \text{graph } G = (V, E), \text{positive natural } k \rangle,\ u)$:

1. `If` u `does not correspond to a set` $S \subseteq V$ `with` $|S| = k$, `reject.`
2. `For each pair of vertices` u, v `in` S:
3. `  If` $\{u, v\} \in E$, `reject.`
4. `Accept.`

We skip the arguments for correctness and running time. ■

Exercise (3SAT is in NP)**.** Show that 3SAT $\in$ NP.

Solution. To show 3SAT is in NP, we need to show that there is a polynomial-time verifier V with the properties stated in Definition (Non-deterministic polynomial time, complexity class NP). We start by presenting V.

def $V(\langle \text{3CNF formula } F \rangle,\ u)$:

1. `If` u `does not correspond to a` $0/1$ `assignment to the variables in` F, `reject.`
2. `Compute the output of the formula` $F(u)$.
3. `If the output is` 0, `reject.`
4. `Else, accept.`

We first show that the verifier V satisfies the two conditions stated in Definition (Non-deterministic polynomial time, complexity class NP). If x is in the language, that means that x corresponds to a valid encoding of a 3CNF formula. Furthermore, there is some $0/1$-assignment to the variables that makes the formula output 1. When u is this $0/1$-assignment, then $|u| = O(n)$ (where n is the length of x), and the verifier accepts the input (x, u). On the other hand, if x is not in the language, then either (i) x is not a valid encoding of a CNF formula in which every clause has exactly 3 literals or (ii) it is a valid encoding but the formula is not satisfiable. In case (i), the verifier rejects (which is done implicitly since the input is not of the correct type). In case (ii), any u that does not correspond to a $0/1$-assignment to the variables makes the verifier reject. Furthermore, any u that does correspond to a $0/1$-assignment to the variables must be such that, with this assignment, the formula evaluates to 0. Therefore, in this case, the verifier again rejects, as desired.

Now we show the verifier is polynomial-time. To check whether u is a valid $0/1$-assignment to the variables takes polynomial time since you just need to check that you are given t bits, where t is the number of variables. The output of the formula can be computed in polynomial time since it takes constant number of steps to compute each clause (and the number of clauses is bounded by the length of the input). ■

Proposition (P is contained in NP)**.** P $\subseteq$ NP.

Proof. Given a language $L \in$ P, we want to show that $L \in$ NP. Since L is in P, we know that there is a polynomial-time decider M that decides L. To show that $L \in$ NP, we need to describe a polynomial-time verifier V that has the properties described in Definition (Non-deterministic polynomial time, complexity class NP). The description of V is as follows.

def $V(x,\ u)$:

1. `Return` $M(x)$.

First, note that since M is a polynomial time decider, the line running $M(x)$ takes polynomial time, and so V is polynomial-time. We now check that V satisfies the two conditions stated in Definition (Non-deterministic polynomial time, complexity class NP). If $x \in L$, then $M(x)$ accepts, so for any u, $V(x,u)$ accepts. For example, $V(x,\epsilon)$ accepts, and clearly $|\epsilon| = 0 \leq |x|$. If $x \notin L$, then $M(x)$ rejects, so no matter what u is, $V(x,u)$ rejects, as desired. This shows that $L \in \mathsf{NP}$. □

Definition (Complexity class EXP)**.** We denote by EXP the set of all languages that can be decided in at most exponential-time, i.e., in time $O(2^{n^C})$ for some constant $C > 0$.

Exercise (NP is contained in EXP)**.** Show that $\mathsf{NP} \subseteq \mathsf{EXP}$.

Solution. To show $\mathsf{NP} \subseteq \mathsf{EXP}$, it suffices to argue that any $L \in \mathsf{NP}$ is also in EXP. So take an arbitrary $L \in \mathsf{NP}$. By the definition of NP we know that there is some polynomial-time verifier TM V and constant $k > 0$ such that for all $x \in \Sigma^*$:

- if $x \in L$, then there is some $u \in \Sigma^*$ with $|u| \leq |x|^k$ such that $V(x,u)$ accepts;
- if $x \notin L$, then for all $u \in \Sigma^*$, $V(x,u)$ rejects.

We construct a decider A for L as follows.

def $A(x)$:

1. `For all` $u \in \Sigma^*$ `with` $|u| \leq |x|^k$`:`
2. `  Run` $V(x,u)$ `and if it accepts, accept.`
3. `Reject`

Correctness: If $x \in L$, then we know that for some $u \in \Sigma^*$ with $|u| \leq |x|^k$, $V(x,u)$ accepts. Therefore A accepts as well. If on the other hand $x \notin L$, then for all $u \in \Sigma^*$, $V(x,u)$ rejects. Therefore A rejects as well.

Running time: The running time of A is $O(2^{n^C})$ for an appropriately chosen constant C. We omit the details of this part. ■

2 NP-complete problems

Note (NP-hardness and NP-completeness)**.** Recall Definition (**??**). We use this definition in this section with $\mathcal{C}$ being NP. However, our notions of NP-hardness and NP-completeness will be restricted to Karp reductions. As mentioned in Note (**??**), hardness and completeness is often defined using Karp reductions. We may explore some of the reasons for this in a homework problem.

Important (Use Karp reductions for NP-hardness)**.** To show that a language is NP-hard, you should be using Karp reductions.

Theorem (Cook-Levin Theorem)**.** $\mathrm{CIRCUIT} - \mathrm{SAT}$ *is* NP*-complete.*

Note. Cook and Levin actually proved that SAT is NP-complete, however, the theorem is often stated using $\mathrm{CIRCUIT} - \mathrm{SAT}$ rather than SAT because the version above is easier to prove.

Important (Showing a language is NP-hard)**.** To show that a language L is NP-hard, by the transitivity of polynomial-time reductions, it suffices to show that $K \leq_m^P L$ for some language K which is known to be NP-hard. In particular, using Theorem (Cook-Levin Theorem), it suffices to show that $\text{CIRCUIT} - \text{SAT} \leq_m^P L$.

Theorem (3COL is NP-complete)**.** *3COL is* NP*-complete.*

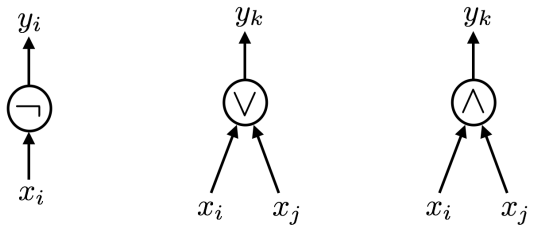
Proof. We have already done all the work to prove that 3COL is NP-complete. First of all, in Proposition (3COL is in NP), we have shown that 3COL $\in$ NP. To show that 3COL is NP-hard, by the transitivity of reductions, it suffices to show that $\text{CIRCUIT} - \text{SAT} \leq_m^P$ 3COL, which we have done in Theorem (**??**). □

Theorem (3SAT is NP-complete)**.** *3SAT is* NP*-complete.*

Proof Sketch. We sketch the main ideas in the proof. To show that 3SAT is NP-complete, we have to show that it is in NP and it is NP-hard. We leave the proof of membership in NP as an exercise.

To show that 3SAT is NP-hard, by the transitivity of reductions, it suffices to show that $\text{CIRCUIT} - \text{SAT} \leq_m^P$ 3SAT. Given an instance of $\text{CIRCUIT} - \text{SAT}$, i.e. a Boolean circuit C, we will construct an instance of 3SAT, i.e. a Boolean CNF formula φ in which every clause has exactly 3 literals. The reduction will be polynomial-time and will be such that C is a Yes instance of $\text{CIRCUIT} - \text{SAT}$ (i.e. C is satisfiable) if and only if φ is a Yes instance of 3SAT (i.e. φ is satisfiable).

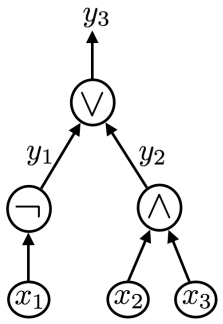
The construction is as follows. A circuit C has three types of gates (excluding the input-gates): NOT, OR, AND.



We will convert each such gate of the circuit C into a 3SAT formula. It is easy to verify that

$$\begin{aligned} y_i = \neg x_i &\iff (x_i \vee y_i) \wedge (\neg x_i \vee \neg y_i), \\ y_k = x_i \vee x_j &\iff (y_k \vee \neg x_i) \wedge (y_k \vee \neg x_j) \wedge (\neg y_k \vee x_i \vee x_j), \\ y_k = x_i \wedge x_j &\iff (\neg y_k \vee x_i) \wedge (\neg y_k \vee x_j) \wedge (y_k \vee \neg x_i \vee \neg x_j). \end{aligned}$$

So the behavior of each gate can be represented using a Boolean formula. As an example, consider the circuit below.



In this case, we would let

$$\varphi_1 = (x_1 \vee y_1) \wedge (\neg x_1 \vee \neg y_1)$$
$$\varphi_2 = (\neg y_2 \vee x_2) \wedge (\neg y_2 \vee x_3) \wedge (y_2 \vee \neg x_2 \vee \neg x_3)$$
$$\varphi_3 = (y_3 \vee \neg y_1) \wedge (y_3 \vee \neg y_2) \wedge (\neg y_3 \vee y_1 \vee y_2),$$

and the Boolean formula equivalent to the circuit would be

$$\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge y_3.$$

This is not quite a 3SAT formula since each clause does not necessarily have exactly 3 literals. However, each clause has at most 3 literals, and every clause in the formula can be converted into a clause with exactly 3 literals by duplicating a literal in the clause if necessary.

This completes the description of how we construct a 3SAT formula from a Boolean circuit. We leave it as an exercise to the reader to verify that C is satisfiable if and only if φ is satisfiable, and that the reduction can be carried out in polynomial time. □

Theorem (CLIQUE is NP-complete)**.** CLIQUE *is* NP*-complete.*

Proof. To show that CLIQUE is NP-complete, we have to show that it is in NP and it is NP-hard. Exercise (CLIQUE is in NP) asks you to show that CLIQUE is in NP, so we will show that CLIQUE is NP-hard by presenting a reduction from 3SAT to CLIQUE.

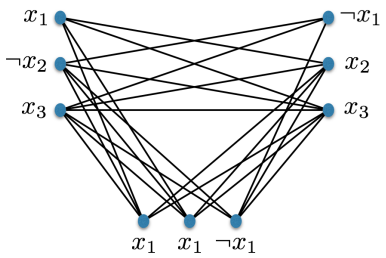
Our reduction will be a Karp reduction. Given an instance of 3SAT, i.e. a Boolean formula φ, we will construct an instance of CLIQUE, $\langle G, k\rangle$ where G is a graph and k is a number, such that φ is a Yes instance of 3SAT (i.e. φ is satisfiable) if and only if $\langle G, k\rangle$ is a Yes instance of CLIQUE (i.e. G contains a k-clique). Furthermore, this construction will be done in polynomial time.

Let

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_m \vee b_m \vee c_m),$$

where each a_i, b_i and c_i is a literal, be an arbitrary 3SAT formula. Notice that φ is satisfiable if and only if there is a truth assignment to the variables so that each clause has at least one literal set to True. From this formula, we build a graph G as follows. For each clause, we create 3 vertices corresponding to the literals of that clause. So in total the graph has $3m$ vertices. We now specify which vertices are connected to each other with an edge. We do *not* put an edge between two vertices if they correspond to the same clause. We do *not* put an edge between x_i and $\neg x_i$ for any i. Every other pair of vertices is connected with an edge. This completes the construction of the graph. We still need to specify k. We set $k = m$.

As an example, if $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_1 \vee \neg x_1)$, then the corresponding graph is as follows:



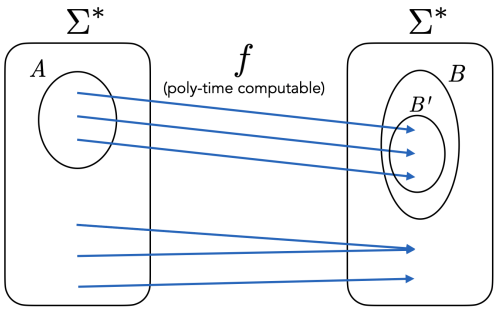
We now prove that φ is satisfiable if and only if G has a clique of size m. If φ is satisfiable, then there is an assignment to the variables such that in each clause, there is at least one literal set to True. We claim that the vertices that correspond to these literals form a clique of size m in G. It is clear that the number of such vertices is m. To see that they form a clique, notice that the only way two of these vertices do not share an edge

is if they correspond to x_i and $\neg x_i$ for some i. But a satisfying assignment cannot assign True to both x_i and $\neg x_i$.

For the other direction, suppose that the constructed graph G has a clique K of size m. Since there are no edges between two literals if they are in the same clause, there must be exactly one vertex from each clause in K. We claim that we can set the literals corresponding to these vertices to True and therefore show that φ is satisfiable. To see this, notice that the only way we could not simultaneously set these literals to True is if two of them correspond to x_i and $\neg x_i$ for some i. But there is no edge between such literals, so they cannot both be in the same clique.

This completes the correctness of the reduction. We still have to argue that it can be done in polynomial time. This is rather straightforward. Creating the vertex set of G is clearly polynomial-time since there is just one vertex for each literal of the Boolean formula. Similarly, the edges can be easily added in polynomial time as there are at most $O(m^2)$ many of them. □

Note. When we want to show that a language B is NP-hard, we take a known NP-hard language A and show a Karp reduction from A to B. The diagram of a typical transformation is as shown below.

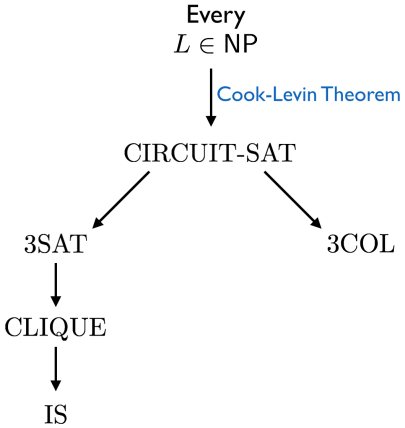


So typically, the transformation does not cover all the elements of B, but only a subset $B' \subset B$. This means that the Karp reduction not only establishes that B is NP-hard, but also that B' is NP-hard.

Theorem (IS is NP-complete)**.** IS *is* NP*-complete.*

Proof. To show that IS is NP-complete, we have to show that it is in NP and it is NP-hard. Exercise (IS is in NP) asks you to show that IS is in NP, so we show that IS is NP-hard. By Theorem (CLIQUE is NP-complete), we know that CLIQUE is NP-hard, and by Theorem (**??**) we know that CLIQUE $\leq_m^P$ IS. By the transitivity of reductions, we conclude that IS is also NP-hard. □

Note (Overview of reductions)**.** The collection of reductions that we have shown can be represented as follows:



Exercise (MSAT is NP-complete)**.** The MSAT problem is defined very similarly to SAT (for the definition of SAT, see Definition (**??**)). In SAT, we output True if and only if the input CNF formula has at least one satisfying truth assignment to the variables. In MSAT, we want to output True if and only if the input CNF formula has at least two distinct satisfying assignments.

Show that MSAT is NP-complete.

Solution. Showing MSAT is in NP: To show that MSAT is in NP, we need to show that there is a polynomial time verifier TM V and $t \in \mathbb{N}$ such that:

1. if $x \in$ MSAT, then there is some u, $|u| \leq |x|^t$, such that $V(x, u)$ accepts;
2. if $x \notin$ MSAT, then for all u, $V(x, u)$ rejects.

We present the verifier.

def $V(\langle$CNF formula $F\rangle,\ u)$:

1. `If` u `does not correspond to two distinct` $0/1$ `assignments to the variables in` F`, reject.`
2. `If both assignments in` u `satisfy` F`, accept.`
3. `Else, reject.`

The verifier is polynomial time: Checking whether u a valid encoding of two distinct $0/1$ assignments to the input variables can be done in polynomial time. Given a $0/1$ assignment to the input variables, we can plug those values into the input CNF formula, and evaluate the output of the formula. This can be done in polynomial time. Since every step can be done in polynomial time, the verifier has polynomial time complexity.

Correctness: If $x \in$ MSAT, then x must a valid encoding of a CNF formula F. Furthermore, F must be such that there are at least two distinct $0/1$ assignments to the input variables that make F evaluate to 1. When u is the encoding of such two distinct $0/1$ assignments, the verifier will accept as desired. Note that clearly for such a u, $|u| \leq O(|F|)$, i.e., the proof is polynomial in the length of F. If $x \notin$ MSAT, then either x is not a valid encoding of a CNF formula (in which case the verifier rejects, which is done implicitly since the input is not of the correct type), or x is a valid encoding of a CNF formula, but there is at most one $0/1$ assignment to the input variables that makes the formula output 1. In the latter case, if u is not the encoding of two distinct $0/1$ assignments, we reject. And if it is, then the last step detects that at least one of these $0/1$ assignments makes the formula output 0, and so the verifier rejects again, as desired.

Showing MSAT is NP-hard: To show that MSAT is NP-hard, we show a Karp reduction from SAT (which we know is NP-hard because we have shown 3SAT is NP-hard) to MSAT. To show the Karp reduction, we need to show that there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in$ SAT if and only if $f(x) \in$ MSAT. Here is the function.

def $f(\langle$CNF formula $F\rangle)$:

1. `Let` z `be a variable name not included in` F`.`
2. $F' = F \wedge (z \vee \neg z)$`.`
3. `Return` $\langle F' \rangle$`.`

Polynomial time: It is pretty clear that each step above can be carried out in polynomial time (with respect to the length of F).

Correctness: Suppose $x \in$ SAT. Then $x = \langle F \rangle$ for some satisfiable CNF formula F. Let $y_1, \ldots, y_n$ be the variables in F. And suppose $y_1 = b_1, \ldots, y_n = b_n$ is a satisfying assignment for F, where $b_i \in \{0, 1\}$ for all i. Then observe that both $y_1 = b_1, \ldots, y_n =$

$b_n, z = 0$ and $y_1 = b_1, \ldots, y_n = b_n, z = 1$ are satisfying assignments for F', and so $\langle F' \rangle \in$ MSAT. For the converse, assume $\langle F' \rangle \in$ MSAT. This means $F' = (F \wedge (z \vee \neg z)) = (F \wedge \text{True})$ is satisfiable, which implies F must satisfiable. So $x = \langle F \rangle \in$ SAT. ■

Exercise (BIN is NP-complete)**.** In the PARTITION problem, we are given n non-negative integers $a_1, a_2, \ldots, a_n$, and we want to output True if and only if there is a way to partition the integers into two parts so that the sum of the two parts are equal. In other words, we want to output True if and only if there is a set $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} a_i = \sum_{j \in \{1,\ldots,n\} \setminus S} a_j$.

In the BIN problem we are given a set of n items with non-negative sizes $s_1, s_2, \ldots, s_n \in \mathbb{N}$ (not necessarily distinct), a capacity $C \geq 0$ (not necessarily an integer), and an integer $k \in \mathbb{N}$. We want to output True if and only if the items can be placed into at most k bins such that the total size of items in each bin does not exceed the capacity C.

Show that BIN is NP-complete assuming PARTITION is NP-hard.

Solution. Showing BIN is in NP: To show that BIN is in NP, we need to show that there is a polynomial time verifier TM V and $t \in \mathbb{N}$ such that:

1. if $x \in$ BIN, then there is some u, $|u| \leq |x|^t$, such that $V(x, u)$ accepts;
2. if $x \notin$ BIN, then for all u, $V(x, u)$ rejects.

We now present the verifier. Below, $s_1, s_2, \ldots, s_n$ and k are non-negative integers and C is a non-negative real.

def $V(\langle s_1, s_2, \ldots, s_n, C, k \rangle,\ u)$:

1. `If` $k > n$`, set` $k = n$`.`
2. `If` u `does not correspond to a partition` $(B_1, B_2, \ldots, B_k)$ `of` $\{1, 2, \ldots, n\}$`, reject.`
3. `If for some` $j \in \{1, 2, \ldots, k\}$`,` $\sum_{i \in B_j} s_i > C$`, reject.`
4. `Else, accept.`

The verifier is polynomial time: (First note that the n in the question does not denote the input length. When we say "polynomial time", it is with respect to the input length, which is $|\langle s_1, s_2, \ldots, s_n, C, k \rangle| + |u|$.) Checking that u is a partition of $\{1, \ldots, n\}$ into k parts, we need to check that there are k sets, their union is $\{1, \ldots, n\}$, and that they are pair-wise disjoint. All of these can be easily checked in polynomial time. In the 3rd step, we implicitly have a loop that repeats $k \leq n$ times. In each iteration, we do at most n additions, which is polynomial time in the input length (even if the s_i's are super big, note that they are part of the input and contribute to the length; addition is polynomial time).

Correctness: If $x \in$ BIN, then first, it must be a valid encoding. Furthermore, there must be a partition $(B_1, \ldots, B_k)$ of $\{1, \ldots, n\}$ such that for each $j = 1, ..., k$, $\sum_{i \in B_j} s_i \leq C$ (this is true by the definition of the problem). When u represents such a partition, the verifier correctly accepts. Such a u has length polynomial in n, and therefore polynomial in the input length, as desired.

If $x \notin$ BIN, then either x is an invalid encoding (in which case the verifier rejects, which is done implicitly since the input is not of the correct type), or x is a valid encoding, but there is no way to partition the elements into k bins $(B_1, \ldots, B_k)$ such that $\sum_{i \in B_j} s_i \leq C$ for all bins. If u does not correspond to a partition $(B_1, \ldots, B_k)$, then we reject. And if it does, we successfully reject on the 3rd step. This completes the correctness argument.

Showing BIN is NP-hard: To show that BIN is NP-hard, we show a Karp reduction from PARTITION to BIN. To show the Karp reduction, we need to show that there is a

polynomial time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in$ PARTITION if and only if $f(x) \in$ BIN.

We now present f. Below $a_1, a_2, \ldots, a_n$ are non-negative integers.

def $f(\langle a_1, a_2, \ldots, a_n \rangle)$:

1. Let $s_i = a_i$ for all i.
2. Let $T = \sum_{i \in \{1,2,\ldots,n\}} s_i$.
3. Let $C = T/2$.
4. Let $k = 2$.
5. Return $\langle s_1, s_2, \ldots, s_n, C, k \rangle$.

Polynomial time: Addition and division are polynomial time operations and we only do these operations polynomially many times. So the whole function is polynomial time computable.

Correctness: If x is in PARTITION, then $x = \langle a_1, a_2, \ldots, a_n \rangle$ for non-negative integers a_i, and there is $S \subset \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = \sum_{j \in \{1,\ldots,n\} \setminus S} a_j$. Let $T = \sum_{i \in \{1,\ldots,n\}} a_i$ be the total sum. Then we must have

$$\sum_{i \in S} a_i = \sum_{j \in \{1,\ldots,n\} \setminus S} a_j = T/2.$$

If we let $B_1 = S$ and $B_2 = \{1, \ldots, n\} \setminus S$, we see that the elements can be partitioned into two bins of capacity $T/2$. Therefore the output $f(x)$ is indeed an element of BIN.

For the other direction, if the output of the reduction $f(x)$ is an element of BIN, then there is a way to partition the elements $s_1 = a_1, \ldots, s_n = a_n$ into $k = 2$ bins B_1 and B_2 ($B_1 \cup B_2 = \{1, \ldots, n\}$, $B_1 \cap B_2 = \varnothing$) of capacity $C = T/2 = \frac{1}{2} \sum_{i \in \{1,\ldots,n\}} s_i$. Since the total sum of the elements is T and each element must be in one of the two bins, the two bins must be completely full, i.e., $\sum_{i \in B_1} a_i = T/2$ and $\sum_{j \in B_2} a_j = T/2$. If we let $S = B_1$, then $\{1, \ldots, n\} \setminus S = B_2$ and so $\sum_{i \in S} a_i = \sum_{j \in \{1,\ldots,n\} \setminus S} a_j$. So $x = \langle a_1, a_2, \ldots, a_n \rangle$ is indeed in PARTITION. ■

3 Check Your Understanding

Problem.

1. Informally, at a high-level, what does it mean for a language to be in NP?
2. Give an example of a language in NP and at a high level explain why it is in NP.
3. What are the things you need to show in order to formally argue that a language L is in NP?
4. Why is the complexity class NP named "Non-deterministic polynomial time"?
5. True or false: Every language in NP is decidable.
6. True or false: Any finite language is in NP.
7. True or false: $\{0,1\}^* \in$ NP.
8. True or false: $\{0^k 1^k : k \in \mathbb{N}\} \in$ NP.
9. Explain why P is contained in NP.
10. True or false: If there is a polynomial-time algorithm for 3COL, then P = NP.

11. True or false: HALTS is NP-complete.
12. Explain why NP is contained in EXP.
13. State the Cook-Levin Theorem and explain its significance.
14. Let $L \subseteq \Sigma^*$, and define its complement $\overline{L} = \{w \in \Sigma^* : w \notin L\}$. Define coNP $= \{L \subseteq \Sigma^* : \overline{L} \in \mathsf{NP}\}$. True or false: If coNP $\neq$ NP, then P $\neq$ NP.
15. True or false: If every language in NP is NP-complete (under Cook reductions) then P $=$ NP.
16. True or false: If P $=$ NP then every language in NP is NP-complete (under Cook reductions).
17. True or false: For any two NP-complete languages A and B, there is a Cook reduction from A to B.
18. True or false: Let $L = \{0^k 1^k : k \in \mathbb{N}\}$. There is a Karp reduction from L to 3COL.
19. Assume P $\neq$ NP. Does 2COL Karp reduce to 3COL?
20. Assume P $\neq$ NP. Does 3COL Karp reduce to 2COL?

4 High-Order Bits

Important.
1. Non-deterministic polynomial time is one of the most important definitions in the course. The formal definition is not easy to understand at first since it has several layers. Take your time to really digest it, and be careful with the order of quantifiers.
2. When we ask you to show that a language is in NP, we expect you to use the formal definition of NP. Your proofs should resemble the proofs presented in this chapter.
3. It is guaranteed that you will see an NP-completeness proof in the homework and exam. In this course, your reductions establishing NP-hardness must be Karp reductions. This chapter contains various such arguments, and we expect your own arguments to follow a similar structure. As usual, do not fall in the trap of memorizing templates. Focus on understanding why the proofs are structured the way they are.