Solutions to Homework 2

Transitivity of Reductions

Problem 1. Let $f, g, h : \{0,1\}^* \to \{0,1\}$ be functions such that $f \leq g$ and $g \leq h$; prove that $f \leq h$. Deduce that if computing f is $\mathsf{NP}-$hard then so is computing g and h. Similarly, deduce that if h can be computed in polynomial time, then so can f and g.

Solution. If $f \leq g$ then there exists an efficient (*i.e.*, n^c time for some constant c) TM M such that for all $\mathbf{x} \in \{0,1\}^*$, $f(\mathbf{x}) = g\big(\mathsf{M}(\mathbf{x})\big)$. Similarly, if $g \leq h$ then there exists an n^d-time TM M' such that for all $\mathbf{x} \in \{0,1\}^*$, $g(\mathbf{x}) = h\big(\mathsf{M}'(\mathbf{x})\big)$. Let M^* be the TM which, on input $\mathbf{x}$, first computes $\mathsf{M}(\mathbf{x})$ and then computes $\mathsf{M}'\big(\mathsf{M}(\mathbf{x})\big)$. Clearly M takes time at most n^{cd} (so M^* is also polynomial time). Note, moreover, that for all $\mathbf{x} \in \{0,1\}^*$,

$$f(\mathbf{x}) = g\big(\mathsf{M}(\mathbf{x})\big) = h\Big(\mathsf{M}'\big(\mathsf{M}(\mathbf{x})\big)\Big) = h\big(\mathsf{M}^*(\mathbf{x})\big),$$

and so $f \leq h$, as desired.

Finally, if f is $\mathsf{NP}-$hard then for all $\varphi \in \mathsf{NP}$, $\varphi \leq f$. By transitivity, $\varphi \leq g$ and $\varphi \leq h$ both hold, and so g and h are both $\mathsf{NP}-$hard as well. Likewise, if $h \in \mathsf{P}$ then there exists an efficient TM M such that for all $\mathbf{x} \in \{0,1\}^*$: $h(\mathbf{x}) = \mathsf{M}(\mathbf{x})$. Let M' be the Turing Machine which, on input $\mathbf{x}$, first runs the reduction guaranteed by $g \leq h$ to get $\mathbf{y}$ and then computes $\mathsf{M}(\mathbf{y})$. Note $g(\mathbf{x}) = \mathsf{M}'(\mathbf{x})$ for all $\mathbf{x}$ and so $g \in \mathsf{P}$ also. The proof that $f \in \mathsf{P}$ is the same except we use the reduction promised by $f \leq h$ instead of the one promised by $g \leq h$.

Some More $\mathsf{NP}-$complete Problems

Problem 2. Prove that three of the following languages are $\mathsf{NP}-$hard.

- **Rules:** For each language you choose, you must give a polynomial time reduction from a known $\mathsf{NP}-$complete language. These can include any of the languages we proved $\mathsf{NP}-$complete in class, as well as any language in the following list (other than the language itself; *i.e.*, you cannot prove L $\mathsf{NP}-$complete by proving $\mathsf{L} \leq \mathsf{L}$).
- $\underline{\mathsf{CIRCUIT\text{-}SAT}} = \{\mathsf{C} \text{ boolean circuit} : \exists\, \mathbf{x} \in \{0,1\}^n \text{ st } \mathsf{C}(\mathbf{x}) = 1\}$.
- $\underline{3\mathsf{SAT} \leq \mathsf{CIRCUIT-SAT}}$**:** Before giving the reduction, we make two comments. First, in class we defined AND and OR gates to have 2 input wires. Note, however, that we can use two such AND gates to build an AND gate with three input wires: $\mathsf{AND}(x_1, x_2, x_3) = \mathsf{AND}\big(\mathsf{AND}(x_1, x_2), x_3\big)$. More generally, we can build an AND gate with k input wires using

$k-1$ AND gates with 2 input wires each. The same is true for OR gates. Therefore, in the reduction below, we allow ourselves, without loss of generality, to output a circuit whose AND and OR gates have large fanin.

Secondly, we allow ourselves to use XOR gates which have two input wires and label their output wire according to the XOR function: $\mathsf{XOR}(a,b)=1$ if and only if exactly one of a and b is labeled 1. This is also without loss of generality since we can build XOR out of AND, OR and NOT gates as follows:

$$\mathsf{XOR}(a,b)=\mathsf{AND}\Big(\mathsf{OR}(a,b),\mathsf{NOT}\big(\mathsf{AND}(a,b)\big)\Big).$$

Given a 3CNF formula Φ with m clauses $\varphi_1,\varphi_2,\ldots,\varphi_m$ over the variables $x_1,\ldots,x_n$, we output the following circuit C whose gates are arranged into four layers so that the input wires to the gates in one layer are the output wires of the gates in the previous layer:

- the bottom layer consists of $2n$ input gates, labeled $g_1^{(1)},\overline{g}_1^{(1)},\ldots,g_n^{(1)},\overline{g}_n^{(1)}$;
- the next layer consists of $m+n$ gates:
  - it has m OR gates labeled $g_1^{(2)},\ldots,g_m^{(2)}$; if φ_j contains the literal x_i (resp. $\overline{x}_i$), then one of the output wires of $g_i^{(1)}$ (resp. $\overline{g}_i^{(1)}$) is an input wire to $g_j^{(2)}$;
  - it has n XOR gates labeled $G_1^{(2)},\ldots,G_n^{(2)}$; for each $i=1,\ldots,n$, there are output wires from $g_i^{(1)}$ and $\overline{g}^{(1)}$ which goes into $G_i^{(2)}$;
- the next layer consists of a single AND gate $g^{(3)}$; every gate in the second layer has a single output wire which is an input wire of $g^{(3)}$;
- the top layer is the output gate $g^{(4)}$ whose only input wire is the output wire of $g^{(3)}$.

Suppose Φ has a satisfying assignment. Label the input gates of C as follows: if $x_i=1$ in the satisfying assignment to Φ, label $g_i^{(1)}$ with 1 and $\overline{g}_i^{(1)}$ with 0; otherwise do the opposite. In this case, the OR gates at the second level are will all be labeled 1 since every clause of Φ is satisfied by the assignment. Moreover, because exactly one of each of x_i and $\overline{x}_i$ is 1, the XOR gates at the second level will also each output 1. Thus, all input wires to the AND gate at the third level are labeled 1, and so the circuit outputs 1.

On the other hand, suppose $\mathsf{C}\in\mathsf{CIRCUIT}-\mathsf{SAT}$, so there is a labeling $\mathbf{v}=(v_1,\overline{v}_1,\ldots,v_n,\overline{v}_n)$ to the input gates of C such that $\mathsf{C}(\mathbf{v})=1$. Consider the assignment to the variables of Φ which sets $x_i=v_i$; we will show this assignment satisfies Φ. As above, $\mathsf{C}(\mathbf{v})=1$ means that all input wires to the AND gate of the third layer are labeled 1. This means that all XOR gates on the second level evaluate to 1 and so have opposite labels on their input bits. This means that $v_i\neq\overline{v}_i$ for all $i=1,\ldots,n$, and so the literal $\overline{x}_i$ gets assigned the value $\overline{v}_i$. Finally, because all of the OR gates on the second level evaluate to 1, they must all have at least one input wire labeled 1. This means that every clause of Φ has at least one true literal, and so Φ is satisfied.

- QUAD-EQ $=\{(\varphi_1,\ldots,\varphi_m)$ satisfiable 3-local quadratic equations over n boolean variables$\}$, where a 3-local quadratic equation over the variables $x_1,\ldots,x_n$ has the form

$$x_ix_j-x_k\equiv 1(\mathrm{mod}\ 2)$$

for some $i, j, k \in \{1, \ldots, n\}$. We say that a sequence of 3-local quadratic equation over n boolean variables is satisfiable if there exists a 0/1 assignment to each of the variables $x_1, \ldots, x_n$ which simultaneously satisfies all the equations.

- $\underline{\mathsf{CIRCUIT-SAT} \leq \mathsf{QUAD-EQ}}$**:** As mentioned in the hint, we assume the circuit consists entirely of NAND gates with two input wires and one output wire. Recall $\mathsf{NAND}(a, b) = 0$ if $a = b = 1$ and is 0 otherwise. Note
  - $\mathsf{NOT}(a) = \mathsf{NAND}(a, a)$;
  - $\mathsf{AND}(a, b) = \mathsf{NOT}\big(\mathsf{NAND}(a, b)\big)$;
  - $\mathsf{OR}(a, b) = \mathsf{NOT}\Big(\mathsf{AND}\big(\mathsf{NOT}(a), \mathsf{NOT}(b)\big)\Big)$.

  So since NAND gates can be used to compute all the other gates, it is without loss of generality to assume our circuits are made up only of NAND gates.

  We now describe the reduction. Given a circuit C with n INPUT gates $g_1^{(\mathsf{INPUT})}, \ldots, g_n^{(\mathsf{INPUT})}$; m NAND gates $g_1^{(\mathsf{NAND})}, \ldots, g_m^{(\mathsf{NAND})}$; and one output gate $g^{(\mathsf{OUTPUT})}$ (we assume the last NAND gate $g_m^{(\mathsf{NAND})}$ has one output wire which is the input wire to the output gate $g^{(\mathsf{OUTPUT})}$), we define $n + m + 1$ variables $x_1, \ldots, x_n, y_1, \ldots, y_m, z$ where the intention is that the x_i's correspond to the INPUT gates, the y_k's correspond to the NAND gates and z corresponds to the OUTPUT gate. Our reduction outputs the following equations over the variables.
  - For every NAND gate $g_k^{(\mathsf{NAND})}$ with input wires w and w', let g and g' respectively be the gates with w and w' as output wires; we output the equation $vv' - y_k \equiv 1 \pmod 2$ where v and v' are the variables which correspond to the gates g and g'. So for example if g is the INPUT gate $g_i^{(\mathsf{INPUT})}$ and g' is the NAND gate $g_j^{(\mathsf{NAND})}$, then $v = x_i$, $v' = y_j$ so the equation is $x_i y_j - y_k \equiv 1 \pmod 2$.
  - Output the equation $y_m z - y_m \equiv 1 \pmod 2$.

  Key Point: $vv' - y_k \equiv 1 \pmod 2$ if and only if $y_k = \mathsf{NAND}(v, v')$.

  We now prove correctness. Suppose C is satisfiable. Then there exists a labeling of the INPUT gates which results in the OUTPUT gate's input wire being labeled 1. We describe an assignment to the variables $x_1, \ldots, x_n, y_1, \ldots, y_m, z$ which satisfies all of the equations. We assign to x_i, the label of the INPUT gate $g_i^{(\mathsf{INPUT})}$; we assign to y_k the label of the output wire of the NAND gate $g_k^{(\mathsf{NAND})}$, and we assign z to 0. Note the equation corresponding to the NAND gate $g_k^{(\mathsf{NAND})}$, $vv' - y_k \equiv 1 \pmod 2$ is satisfied since by definition $y_k = \mathsf{NAND}(v, v')$. Finally, since C is satisfied by the labeling, $y_m = 1$ and so the output equation is also satisfied:

$$y_m z - y_m = 1 \cdot 0 - 1 = -1 \equiv 1 \pmod 2.$$

  On the other hand, if C is not satisfiable, then all labels to the INPUT gates result in the input wire of the OUTPUT gate being labeled 0. It follows that no matter what values we assign to the x_i's, if every NAND equation is satisfied (*i.e.*, if $y_k = \mathsf{NAND}(v, v')$ for all k) then $y_m = 0$. However, this implies that the output equation cannot be satisfied: $y_m z - y_m = y_m(z - 1) = 0 \not\equiv 1 \pmod 2$. Therefore, the equations are not satisfiable.

- $\underline{\mathsf{VERTEX\text{-}COVER}}$
$= \{(\mathsf{G}, k) : \exists\ S \subset \mathsf{V} \text{ st } |S| = k \text{ and } \forall\ (v, v') \in \mathsf{E}, \text{ either } v \in S \text{ or } v' \in S\}$.

- $\underline{\mathsf{INDSET} \leq \mathsf{VERTEX-COVER}}$**:** The reduction takes (G, k) (a graph $G = (V, E)$ and integer k) and outputs $(G, n - k)$ where $n = |V|$. The key point is that $S \subset V$ is an independent set if and only if $\overline{S} \subset V$ (*i.e.*, the complement of S) is a vertex cover. Indeed, if S is an independent set then there are no edges in G which have both endpoints in S. In other words, for every edge in E, at least one endpoint is in $\overline{S}$, and so $\overline{S}$ is by definition a vertex cover (the other direction is the same). Correctness of the reduction follows immediately since G has an independent set of size k if and only if it has a vertex cover of size $n - k$.

- $\underline{\mathsf{dHAMPATH}}$ $= \{\mathsf{G} \text{ directed graph} : \exists \text{ Hamiltonian cycle}\}$, where a Hamiltonian cycle in a graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ is a list of vertices $v_0, v_1, \ldots, v_n$ such that $v_0 = v_n$ and each other vertex in V appears exactly once in the list and moreover $(v_{i-1}, v_i) \in \mathsf{E}$ for all $i = 1, 2, \ldots, n$.

- $\underline{\mathsf{3SAT} \leq \mathsf{dHAMPATH}}$**:** The reduction takes a 3CNF formula Φ with m clauses $\varphi_1, \ldots, \varphi_m$ and n variables $x_1, \ldots, x_n$ and outputs a graph with $m + 2nk + 3$ vertices where $k \geq 2m$ is some integer. The intuition of the reduction is that each variable of Φ corresponds to a chain of $2k$ vertices, and a 1/0 assignment to the variable corresponds to traversing the chain from left-to-right/right-to-left. Then there is one vertex for each clause, connected to the chains of its literals in such a way so that the clause vertex can only be reached in a cycle if one of its literals is traversing its chain in the correct way. We proceed formally. The vertices are arranged and connected as follows:

  - A terminal vertex v_{term} that will be the start and end point of our cycle.
  - A chain of k vertices for each variable; for the variable x_i we label the vertices in the chain $v_{i,1}, \ldots, v_{i,k}$. We connect all adjacent vertices in each chain with edges in both directions: $(v_{i,j}, v_{i,j+1}), (v_{i,j+1}, v_{i,j}) \in E$ for all $j = 1, \ldots, k - 1$. We connect the ends and beginnings of successive chains

    $$(v_{i,1}, v_{i+1,1}), (v_{i,1}, v_{i+1,k}), (v_{i,k}, v_{i+1,1}), (v_{i,k}, v_{i+1,k}) \in E$$

    for $i = 1, \ldots, n - 1$. Finally, we connect the terminal vertex to the ends of the first chain, and the ends of the last chain to the terminal vertex:

    $$(v_{\mathsf{term}}, v_{1,1}), (v_{\mathsf{term}}, v_{1,k}), (v_{n,k}, v_{\mathsf{term}}), (v_{n,1}, v_{\mathsf{term}}) \in E.$$

  - One vertex for each clause; for clause φ_α we denote the corresponding vertex w_α for $\alpha = 1, \ldots, m$. We connect the w_α to the chains of the literals they contain as follows: if φ_α contain x_i, then some $j \in \{1, \ldots, k\}$ is chosen so that neither of the vertices $v_{i,j}, v_{i,j+1}$ are connected to any other w_β another clause φ_β. Then if the literal x_i is in φ_α then the edges $(v_{i,j}, w_\alpha)$ and $(w_\alpha, v_{i,j+1})$ are added to E; if $\overline{x_i}$ is in φ_α then the edges $(v_{i,j+1}, w_\alpha)$ and $(w_\alpha, v_{i,j})$ are added to E.

  Some intuition is in order. Suppose we have a Hamiltonian path in the above graph, and assume without loss of generality that it starts and ends at v_{term}. This path will traverse

the n chains in order starting with the chain corresponding to x_1, ending with the chain ending with x_n. For each i it either goes from $v_{i,1}$ to $v_{i,k}$ in order, or it goes in "reverse order" starting at $v_{i,k}$ to $v_{i,1}$. Our intention is that whether the path traverses the $i-$th chain in order or in reverse order should correspond to whether the variable x_i is assigned 1 or 0. Finally, we have connected the clause vertices to the chains in such a way so that the path can only deviate from the chain to pass through the clause vertex if it is traversing the chain in the direction consistent with the clause literal being true. Thus, the graph can only have a Hamiltonian path if every clause contains at least one true literal. We now proceed formally.

First, assume that Φ is satisfiable. Then there is an assignment to the x_i such that each clause φ_α contains a true literal. Consider the path which starts at v_{term}, then traverses the successive chains in order where it traverses the $i-$th chain in the direction corresponding to the assignment of x_i (described next), and then ends back at v_{term}. If x_i is assigned 1, then the path traverses the $i-$th chain in order: $(v_{i,1}, v_{i,2}, \ldots, v_{i,k})$. If x_i is assigned 0 then the path traverses in the reverse order: $(v_{i,k}, \ldots, v_{i,1})$. Finally, since the assignment satisfies Φ, each clause φ_α contains a true literal, say either x_i or $\overline{x}_i$. If x_i is true then we are traversing the $i-$th chain in order, and so our path deviates from and returns to the $i-$th chain: $(v_{i,j}, w_\alpha), (w_\alpha, v_{i,j+1})$ where j is the $j \in \{1, \ldots, k\}$ chosen during the reduction. Likewise, if $\overline{x}_i$ is true then we are traversing the $i-$th chain in reverse order, and so our path deviates from and returns as follows: $(v_{i,j+1}, w_\alpha), (w_\alpha, v_{i,j})$. So we see that the graph has a Hamiltonian cycle.

On the other hand, suppose the graph has a Hamiltonian cycle. Without loss of generality, assume the cycle and ends at v_{term}. We claim that the path can only pass through the clause vertex w_α in one of the following two ways:

- $(v_{i,j}, w_\alpha, v_{i,j+1})$ if φ_α contains the literal x_i; or
- $(v_{i,j+1}, w_\alpha, v_{i,j})$ if φ_α contains the literal $\overline{x}_i$.

In other words, the only way the path can pass through w_α is by departing from and returning to the same chain. This is because, if the path goes from $v_{i,j}$ to w_α to $v_{i',j'}$ for some (i', j') with $i' \neq i$, then the path must arrive to $v_{i,j+1}$ from $v_{i,j+2}$ (since all other neighbors of $v_{i,j+1}$) have already been traversed), at which point it will be stuck.

It follows that the path must traverse the chains in order from 1 to n where each chain is either traversed in order or in reverse order deviating as described above to pass through the clause vertices. Because the path passes through all clause variables, it follows that for all α, either:

- φ_α contains x_i and the path traverses the $i-$th chain in order; or
- φ_α contains $\overline{x}_i$ and the path traverses the $i-$th chain in reverse order.

Therefore, if we assign 1 to the x_i for which the path traverses the $i-$th chain in order and 0 to the x_i for which the path traverses the $i-$th chain in reverse order, then every clause is guaranteed to have a true literal. Correctness of our reduction follows.

- $\underline{\mathsf{HAMPATH}} = \{\mathsf{G} \text{ undirected graph} : \exists \text{ Hamiltonian cycle}\}$, where a Hamiltonian cycle in a graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ is a list of vertices $v_0, v_1, \ldots, v_n$ such that $v_0 = v_n$ and each other

vertex in V appears exactly once in the list and moreover $(v_{i-1}, v_i) \in \mathsf{E}$ for all $i = 1, 2, \ldots, n$.

- $\underline{\mathsf{dHAMPATH} \leq \mathsf{HAMPATH}}$**:** The reduction takes a directed graph $G = (V, E)$ as input and outputs an undirected graph $G' = (V', E')$ as follows:

  - G' has three vertices, (v_1, v_2, v_3), for every vertex v of G. These three vertices are connected in a line: that is, $(v_1, v_2), (v_2, v_3) \in E'$ for every $v \in V$.
  - For every edge $(v, w) \in E$, the edge (v_3, w_1) is added to E'.

  The intuition is that because the middle vertices v_2 are connected only to v_1 and v_3, any cycle in G' must progress through the vertices (v_1, v_2, v_3) by going into v_1 from another vertex, then going to v_2 then to v_3 and out to another vertex (it could also do this in reverse order).

  Suppose G has a cycle, say C. We construct a cycle C' in G' by turning every edge of C into a path of length three in C': the edge (v, w) in C corresponds to the path (v_2, v_3, w_1, w_2) in C'. Note that C' is a cycle since it starts and ends at v_2 (v_2 is the starting and ending node of C), and it passes through every vertex of G' once (since C passes through every vertex of G once).

  Conversely, suppose G' has a cycle, say C'. Assume without loss of generality that C' starts and ends at v_2 and first travels from v_2 to v_3 (C' can be rearranged to make this the case). Then since v_3 is connected only to v_2 and w_1 for any $w \in V$ such that $(v, w) \in E$, C' must go from v_3 to some w_1. Note that C' must next travel from w_1 to w_2 since if not then later when C' passes from w_3 (w_2's only other neighbor) to w_2, it will be stuck. Thus, paths of length 3 in C' have the form (v_2, v_3, w_1, w_2) and so correspond to edges (v, w) in G. Thus, C' gives a cycle C in G.

- $\underline{\mathsf{IPROG}} = \{S \text{ integer linear system of inequalities} : \exists \text{ satisfying assignment}\}$, where an integer linear system of inequalities, S, over the variables $\{x_1, \ldots, x_n\}$ is a set of inequalities of the form $a_1 x_1 + \cdots + a_n x_n \geq b$ where $a_i, b \in \mathbb{Z}$; and we say S has a satisfying assignment if there exist integer values for the x_i such that each inequality in S is satisfied.

- $\underline{\mathsf{3SAT} \leq \mathsf{IPROG}}$**:** The reduction takes a 3CNF formula Φ over variables $x_1, \ldots, x_n$ with m clauses $\varphi_1, \ldots, \varphi_m$ and outputs a set S of inequalities over the integer variables $\{x_1, \ldots, x_n\}$ consisting of the following two types of inequalities:

  1. For each $i = 1, \ldots, n$: $0 \leq x_i \leq 1$ (formally $x_i \geq 0$ and $-x_i \geq -1$);
  2. For each $\alpha = 1, \ldots, m$: if $\varphi_\alpha = x_i \vee \overline{x_j} \vee \overline{x_k}$ then include $x_i + (1 - x_j) + (1 - x_k) \geq 1$.

  Note that a boolean assignment to the $\{x_1, \ldots, x_n\}$ that satisfies Φ satisfies all of the above inequalities: the inequalities of type 1 are satisfied because the x_i are boolean; the inequalities of type 2 are satisfied because each clause has a true literal. Conversely, any integer assignment to the $\{x_1, \ldots, x_n\}$ that satisfies all of the above inequalities must be boolean (since satisfies type 1 inequalities) and must be such that each clause of φ has a true literal (since satisfies type 2 inequalities).

2SAT $\in$ P

Problem 3. Describe a polynomial time algorithm which decides 2SAT. Deduce that 2SAT $\in$ P.

Solution. Given a 2CNF formula φ over n variables $\{x_1, \ldots, x_n\}$, construct a directed graph $G_\varphi = (V, E)$ as follows. G_φ has $2n$ vertices: $V = \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$, one vertex in G_φ for each of the $2n$ possible literals of φ. If the clause $x_i \vee \overline{x}_j$ appears in φ then the edges $(\overline{x}_i, \overline{x}_j)$ and (x_j, x_i) are added to E. Clearly the graph G_φ can be constructed efficiently. The intuition is that $(v, v') \in E$ stipulates: if the literal v is TRUE then so is the literal v'. Formally, we prove the following claim.

Claim. φ is satisfiable if and only if:

$$\nexists\ i \in \{1, \ldots, n\} \text{ st } (G_\varphi, x_i, \overline{x}_i) \in \mathsf{PATH} \text{ and } (G_\varphi, \overline{x}_i, x_i) \in \mathsf{PATH},$$

where $(G_\varphi, x_i, \overline{x}_i) \in$ PATH means that there is a path in G_φ from the vertex corresponding to x_i to the vertex corresponding to $\overline{x}_i$. Since PATH $\in$ P (it is efficiently solved by Dijkstra's algorithm), the Claim ensures that the following efficient algorithm solves 2SAT.

1. Given a 2CNF formula φ, compute $G_\varphi = f(\varphi)$.
2. For each $i = 1, \ldots, n$:
   - check (using Dijkstra) whether $(G_\varphi, x_i, \overline{x}_i) \in$ PATH or $(G_\varphi, \overline{x}_i, x_i) \in$ PATH; if both checks return "yes", halt and output 0, otherwise continue.
3. Output 1 and halt.

Note that the algorithm outputs 1 if and only if there does not exist $i \in \{1, \ldots, n\}$ such that $(G_\varphi, x_i, \overline{x}_i) \in$ PATH and $(G_\varphi, \overline{x}_i, x_i) \in$ PATH which is the case if and only if $\varphi \in$ 2SAT by the claim.

Proof of Claim. Suppose φ is satisfiable, and consider a satisfying assignment to the variables of φ. For each i, let $y_i \in \{x_i, \overline{x}_i\}$ be the literal which is assigned TRUE and let $\overline{y}_i \in \{x_i, \overline{x}_i\}$ be the literal assigned FALSE. We claim that $(G_\varphi, y_i, \overline{y}_i) \notin$ PATH. To see this, suppose y_i and $\overline{y}_i$ are connected in G_φ via the path $(y_i, z_1, \ldots, z_k, \overline{y}_i)$. These egdes arise in G_φ due to the following $(k+1)$ clauses being in φ:

$$\overline{y}_i \vee z_1;\ \overline{z}_1 \vee z_2; \cdots \overline{z}_{k-1} \vee z_k;\ \overline{z}_k \vee \overline{y}_i.$$

These clauses cannot all be satisfied since either z_1 is FALSE in which case the first clause is not satisfied; or z_k is TRUE in which case the last clause is not satisfied; or there exists some $i = 1, \ldots, k-1$ such that z_i is TRUE but z_{i+1} is FALSE, in which case the clause $\overline{z}_i \vee z_{i+1}$ is not satisfied.

Conversely, suppose for all $i \in \{1, \ldots, n\}$ either $(G_\varphi, x_i, \overline{x}_i)$ or $(G_\varphi, \overline{x}_i, x_i) \notin$ PATH. First, let us extend G_φ to a new graph G'_φ by adding a few edges. Specifically, for every $i \in \{1, \ldots, n\}$ such that neither of $(G_\varphi, x_i, \overline{x}_i)$ or $(G_\varphi, \overline{x}_i, x_i)$ is in PATH, let us add the edge $(x_i, \overline{x}_i)$ to E. Now we are guaranteed that for each i, at least one of $(G'_\varphi, x_i, \overline{x}_i)$ and $(G'_\varphi, \overline{x}_i, x_i)$ is in PATH. It can be shown that actually, exactly one of the two pairs is in PATH for each i. To see this, suppose

adding the edge $(x_i, \overline{x}_i)$ to E caused some j to be so that $(G'_\varphi, x_j, \overline{x}_j)$ and $(G'_\varphi, \overline{x}_j, x_j) \in \mathsf{PATH}$. One possibility (the others are similar) is that there were paths in G'_φ from x_j to x_i and $\overline{x}_i$ to $\overline{x}_j$ as well as from $\overline{x}_j$ to x_j. This implies that $(G'_\varphi, \overline{x}_i, x_i) \in \mathsf{PATH}$ already (since we can compose the paths from $\overline{x}_i$ to $\overline{x}_j$ to x_j to x_i) and so $(x_i, \overline{x}_i)$ would not have been added to E.

Now that we have G'_φ for which it holds that for all $i \in \{1, \ldots, n\}$ exactly one of $(G'_\varphi, x_i, \overline{x}_i)$, $(G'_\varphi, \overline{x}_i, x_i) \in \mathsf{PATH}$, we build an assignment to the variables of φ as follows: if $(G'_\varphi, x_i, \overline{x}_i) \in \mathsf{PATH}$ set $x_i = \mathsf{FALSE}$, otherwise set $x_i = \mathsf{TRUE}$. We show that no edge $(v, v') \in E$ of G'_φ has been assigned $(v, v') = (\mathsf{TRUE}, \mathsf{FALSE})$. It follows that every clause of φ has at least one TRUE literal (recall every clause $\overline{v} \vee v'$ of φ corresponds to an edge (v, v') of G'_φ). Suppose (v, v') has been assigned $(v, v') = (\mathsf{TRUE}, \mathsf{FALSE})$. Since $v' = \mathsf{FALSE}$ it must be that $(G'_\varphi, v', \overline{v}') \in \mathsf{PATH}$. Since $v = \mathsf{TRUE}$ it must be that $(G'_\varphi, \overline{v}, v) \in \mathsf{PATH}$. Since $(v, v') \in E$ if and only if $(\overline{v}', \overline{v}) \in E$ (since both edges correspond to the clause $\overline{v} \vee v'$), we have paths in G'_φ from $\overline{v}'$ to $\overline{v}$, $\overline{v}$ to v, v to v', and hence $(G'_\varphi, \overline{v}', v') \in \mathsf{PATH}$ and $(G'_\varphi, v', \overline{v}') \in \mathsf{PATH}$, a contradiction. □

EXP vs. NEXP

The complexity classes EXP and NEXP are the analogues of P and NP but for exponential time, rather than polynomial time. So specifically,

$$\mathsf{EXP} = \bigcup_{c \geq 1} \mathsf{DTIME}(2^{n^c}); \ \mathsf{NEXP} = \bigcup_{c \geq 1} \mathsf{NTIME}(2^{n^c}).$$

Problem 4. Prove that if $\mathsf{P} = \mathsf{NP}$ then $\mathsf{EXP} = \mathsf{NEXP}$.

Solution. Suppose $\mathsf{P} = \mathsf{NP}$. Let $f \in \mathsf{NEXP}$ be a function. So there exists a $2^{n^c}-$time TM M such that for all $\mathbf{x} \in \{0,1\}^*$:

$$f(\mathbf{x}) = 1 \iff \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}(\mathbf{x}, \mathbf{w}) = 1.$$

Define the *padded version of* M, $\mathsf{M}_{\mathsf{pad}}$ to be the TM which takes $(\mathbf{x}, \mathbf{w}, 1^{2^{|\mathbf{x}|^c}})$ as input and simply ignores the third coordinate and outputs $\mathsf{M}(\mathbf{x}, \mathbf{w})$. Note M and $\mathsf{M}_{\mathsf{pad}}$ give the same output, but because the input to $\mathsf{M}_{\mathsf{pad}}$ is so much longer, $\mathsf{M}_{\mathsf{pad}}$ runs in time that is linear in its input. Let $g(\mathbf{x}, 1^{2^{|\mathbf{x}|^c}})$ be the function which ignores the second input and just outputs $f(\mathbf{x})$. Note $g \in \mathsf{NP}$ since

$$g(\mathbf{x}, 1^{2^{|\mathbf{x}|^c}}) = 1 \iff \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}_{\mathsf{pad}}(\mathbf{x}, \mathbf{w}, 1^{2^{|\mathbf{x}|^c}}) = 1,$$

and $\mathsf{M}_{\mathsf{pad}}$ is a linear time TM. Since $\mathsf{P} = \mathsf{NP}$, there exists an efficient (say time n^d) TM $\mathsf{M}^*_{\mathsf{pad}}$ which computes g; *i.e.*, $g(\mathbf{x}, 1^{2^{|\mathbf{x}|^c}}) = 1$ if and only if $\mathsf{M}^*_{\mathsf{pad}}(\mathbf{x}, 1^{2^{|\mathbf{x}|^c}}) = 1$. Finally, let M^* be the TM which takes input $\mathbf{x}$, writes down the string $(\mathbf{x}, 1^{2^{|\mathbf{x}|^c}})$ and then passes $(\mathbf{x}, 1^{2^{|\mathbf{x}|^c}})$ to $\mathsf{M}^*_{\mathsf{pad}}$ and outputs whatever $\mathsf{M}^*_{\mathsf{pad}}$ returns. Note that M^* runs in time $2^{\mathcal{O}(n^c)}$, and moreover that

$$\begin{aligned} \mathsf{M}^*(\mathbf{x}) = 1 &\iff \mathsf{M}^*_{\mathsf{pad}}(\mathbf{x}, 1^{2^{|\mathbf{x}|^c}}) = 1 \iff \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}_{\mathsf{pad}}(\mathbf{x}, \mathbf{w}, 1^{2^{|\mathbf{x}|^c}}) = 1 \\ &\iff \exists \mathbf{w} \in \{0,1\}^* \text{ s.t. } \mathsf{M}(\mathbf{x}, \mathbf{w}) = 1 \iff f(\mathbf{x}) = 1. \end{aligned}$$

Thus, M^* is a $2^{\mathcal{O}(n^c)}-$time TM which computes f, thus $f \in \mathsf{EXP}$.

A Search-to-Decision Reduction for SAT

So far in class, we have been interested in Turing Machines which *decide* SAT and other languages in NP (*i.e.*, M takes a formula Φ and outputs 1 or 0 based on whether Φ is satisfiable or not; we say M *decides* SAT or equivalently that M solves the *decision version* of SAT). One could ask for more. We might want M to output a satisfying assignment for Φ if $\Phi \in$ SAT. In this case we say that M solves the *search version* of SAT.

Problem 5. Suppose we have oracle access to the decision function $f_{\mathsf{dSAT}} : \{\mathsf{CNFs}\} \to \{0,1\}$ where $f_{\mathsf{dSAT}}(\Phi) = 1$ if and only if Φ is satisfiable. Describe a deterministic polynomnial time $f_{\mathsf{dSAT}}-$oracle algorithm which takes as input a CNF formula over n boolean variables $\{x_1, \ldots, x_n\}$ and by making $2n-1$ oracle calls to f_{dSAT} outputs a satisfying assignment to the $\{x_i\}$ whenever there is one. Deduce that solving the search version of SAT is no harder than solving the decision version. Such results are called *search to decision* reductions.

Solution. Let Φ be CNF formula over the variables $\{x_1, \ldots, x_n\}$. For a bit $b_1 \in \{0,1\}$, we denote by $\Phi|_{x_1=b_1}$ the CNF formula created from Φ by assigning the variable x_1 to b_1. Similarly, for any set $S \subset \{1, \ldots, n\}$ and corresponding bit string $\mathbf{b}_S = (b_i)_{i \in S}$, we let $\Phi|_{\mathbf{x}_S = \mathbf{b}_S}$ be the CNF formula obtained from Φ by setting $x_i = b_i$ for all $i \in S$. Note that Φ is satisfiable if and only if at least one of $\Phi|_{x_1=0}$ and $\Phi|_{x_1=1}$ is satisfiable. Similarly, for any $S \subset \{1, \ldots, n\}$, Φ is satisfiable if and only if there exists $\mathbf{b}_S \in \{0,1\}^{|S|}$ such that $\Phi|_{\mathbf{x}_S = \mathbf{b}_S}$ is satisfiable. With this notation and observation in place, we now describe our algorithm.

Our algorithm $\mathcal{A}$ is given a CNF formula over n variables $\{x_1, \ldots, x_n\}$ and has oracle access to the decision function f_{dSAT} which decides the decision form of SAT. $\mathcal{A}$ proceeds recursively as follows:

1. Initialize an 'empty assignment', $\mathsf{L} = \emptyset$ to return. As the algorithm progresses, we will add equations of the form '$x_i = b$' to L so that by the end of the algorithm L will constitute an assignment to all the variables of Φ.

2. If $n = 1$, then check both possible assignments $x_1 = 0$ and $x_1 = 1$ by hand; if one of these, say $x_1 = b$, satisfies Φ add '$x_1 = b$' to L and output L; otherwise halt with no output. Moving forward we assume $n > 1$.

3. Compute the CNF formulas $\Phi|_{x_n=0}$ and $\Phi|_{x_n=1}$ and send each to the f_{dSAT} oracle; note that both $\Phi|_{x_n=0}$ and $\Phi_{x_n=1}$ are CNF formulas over $n-1$ variables: $\{x_1, \ldots, x_{n-1}\}$. If the oracle responds 0 for both (indicating that neither has a satisfying assignment) then halt and give no output; otherwise, redefine Φ to be (one of) the $\Phi|_{x_n=b}$ for which the oracle responded positively, add '$x_n = b$' to L and redefine $n = n - 1$. Return to step 2 with the new CNF formula Φ and repeat.

Note that each time Step 3 is run, the number of variables of Φ is reduced by 1, and so the loop will repeat at most n times; each time through the loop (except the final time), the algorithm makes 2 oracles calls, therefore $\mathcal{A}$ calls the f_{dSAT} oracle $2n-2$ times. Finally, if $\mathcal{A}(\Phi)$ outputs L, let $\mathbf{b} \in \{0,1\}^n$ be such that L contains the equations '$x_i = b_i$' for all $i = 1, \ldots, n$. Let $S_i = \{i+1, \ldots, n\} \subset \{1, \ldots, n\}$. We have for all $i = 1, \ldots, n$: $x_j = b_j$ for $j \leq i$ is a satisfying assignment for $\Phi|_{\mathbf{x}_{S_i} = \mathbf{b}_{S_i}}$. In particular, $\mathbf{x} = \mathbf{b}$ is a satisfying assignment for $\Phi = \Phi_{\mathbf{x}_{S_n} = \mathbf{b}_{S_n}}$.