1 Reductions, Continued

Example 1 Undirected Hamiltonian Path (UHP) and Undirected Hamiltonian Circuit (UHC) are the same problems as the corresponding problems for directed graphs, but the new problems are for undirected graphs.

Claim 2 Given a polynomial time algorithm for Undirected Hamiltonian Path (UHP) there is a polynomial time algorithm for Directed Hamiltonian Path.

Proof: Let \((G, s, t)\) be the input to the Directed Hamiltonian Path algorithm, \(A_{\text{DHP}}\). The algorithm proceeds as follows.

1. \(A_{\text{DHP}}\) builds the triple \((H, p, q)\), where \(H\) is an undirected graph, and \(p\) and \(q\) are vertices in \(H\) with the property that
   \[(G, s, t) \in \text{DHP} \iff (H, p, q) \in \text{UHP}.\]

2. It runs \(A_{\text{UHP}}(H, p, q)\).

3. It reports the answer given by \(A_{\text{UHP}}(H, p, q)\).

   Let \(G = (V, E)\) and \(H = (F, W)\). Let \(n = |V|\). \(H\) is constructed as follows. For each vertex \(v \in V\), \(H\) has 3 vertices \(v_{\text{in}}, v_{\text{mid}}, v_{\text{out}}\), plus the edges \((v_{\text{in}}, v_{\text{mid}}), (v_{\text{mid}}, v_{\text{out}})\). And for each edge \((v, w) \in E\), \(H\) receives edge \((v_{\text{out}}, w_{\text{in}})\). See Figure 1 for an illustration. Then \(A_{\text{DHP}}\) sets \(p = s_{\text{in}}\) and \(q = t_{\text{out}}\).

4. We argue that \(G\) has a Hamiltonian Path from \(s\) to \(t\) exactly if \(H\) has a Hamiltonian Path from \(p\) to \(q\).

   First, suppose that \(G\) has a Hamiltonian Path from \(s\) to \(t\). Suppose that the path is \(u_1, u_2, \ldots, u_n\), where \(s = u_1, t = u_n\), and \(u_1, u_2, \ldots, u_n\) is a permutation of the vertices in \(V\). Then the following path is a Hamiltonian Path from \(p\) to \(q\) in \(H\).

   \[u_1^{\text{in}}, u_1^{\text{mid}}, u_1^{\text{out}}, u_2^{\text{in}}, u_2^{\text{mid}}, u_2^{\text{out}}, \ldots, u_n^{\text{in}}, u_n^{\text{mid}}, u_n^{\text{out}}.\]
Next, suppose that \( H \) has a Hamiltonian Path \( P \) from \( p \) to \( q \). For \( P \) to go through vertex \( v_{\text{mid}} \), \( P \) must include edges \( (v_{\text{in}}, v_{\text{mid}}) \) and \( (v_{\text{mid}}, v_{\text{out}}) \). Thus the path has the form
\[
v_{1, \text{in}}, v_{1, \text{mid}}, v_{1, \text{out}}, v_{2, \text{in}}, v_{2, \text{mid}}, \cdots, v_{n, \text{in}}, v_{n, \text{mid}}, v_{n, \text{out}},
\]
where \( |V| = n \), \( s = v_1 \), \( t = v_n \), and \( v_1, v_2, \cdots, v_n \) is a permutation of the vertices in \( V \). Then \( v_1, v_2, \cdots, v_n \) is a Hamiltonian Path from \( s \) to \( t \) in \( G \).

This shows the claim.

5. Clearly \( A_{\text{DHP}} \) runs in polynomial time if \( A_{\text{UHP}} \) runs in polynomial time. \( \square \)

**Example 3** Vertex Cover (VC).

**Input:** \((G, k)\), where \( G = (V, E) \) is an undirected graph and \( k \) is an integer.

**Question:** Does \( G \) have a vertex cover of size \( k \), that is a subset \( U \subseteq V \) of size at most \( k \) with the property that every edge in \( E \) has at least one endpoint in \( E \)?

**Claim 4** Given a polynomial time algorithm for Independent Set (IS) there is a polynomial time algorithm for Vertex Cover.

**Proof:** This is immediate from the following observation. If \( I \) is an independent set of \( G \) then \( V - I \) is a vertex cover, and conversely. For if \( I \) is an independent set, then as no edge joins pairs of vertices in \( I \), any edge has at most one endpoint in \( I \), and hence at least one endpoint in \( V - I \). Conversely, if \( V - I \) is a vertex cover, then any edge has at least one endpoint in \( V - I \), and so there is no edge with two endpoints in \( I \), meaning that \( I \) is an independent set.

We can conclude that \((G, k) \in \text{VC} \iff (G, |V| - k) \in \text{IS}\).

We leave the details of creating the algorithm for Vertex Cover to the reader. \( \square \)

**Example 5** 3-SAT.

**Input:** \( F \), a CNF Boolean Formula in which each clause has at most 3 variables.

**Question:** Does \( F \) have a satisfying assignment (of Boolean values to its variables)?
Claim 6 Given a polynomial time algorithm for Independent Set (IS) there is a polynomial time algorithm for 3-SAT.

Proof:
Let $F$ be the input to the 3-SAT algorithm, $A_{3-SAT}$. The algorithm proceeds as follows.

1. $A_{3-SAT}$ computes $(G, k)$, where $G$ is a graph and $k$ is an integer, with the property that
   \[ F \in 3\text{-SAT} \iff G \in IS. \]

2. It runs $A_{IS}(G, k)$.

3. It reports the answer given by $A_{IS}(G, k)$.

   Let $F = C_1 \land C_2 \land \cdots \land C_l$, and suppose that $F$ has variables $x_1, x_2, \ldots, x_n$. $G$ receives the following vertices. For each variable $x$ appearing in $F$ it has vertices $v_x$ and $v_{\overline{x}}$ connected together. For each clause $C_i = a \lor b \lor c$ it has vertices $u_{a}^i, u_{b}^i, u_{c}^i$ connected in a triangle (if the clause is $C_i = a \lor b$ it has vertices $u_{a}^i, u_{b}^i$ joined by an edge, and if $C_i = a$ it has the vertex $u_{a}^i$ alone). In addition, each vertex $u_{x}^i$ is joined to vertex $v_x$ and each vertex $u_{\overline{x}}^i$ is joined to $v_{\overline{x}}$. See Figure 3 for an example. Finally, $A_{3-SAT}$ sets $k = n + l$.

4. We argue that $F$ has a satisfying assignment exactly if $G$ has an independent set $I$ of size $n + l$.

   Suppose that $F$ has a satisfying assignment $\sigma$. If $x_i = \text{True}$ in $\sigma$, then $v_{x_i}$ is put in the independent set $I$, while if $x_i = \text{False}$ in $\sigma$, then $v_{\overline{x}_i}$ is put in $I$. This adds $n$ vertices to $I$, and none of these vertices have connecting edges. In addition, for each clause $C_i = a \lor b \lor c$, at least one of its literals must evaluate to True; without loss of generality, let $a$ be this True literal. Then $u_{a}^i$ is added to $I$. This adds another $l$ vertices to $I$, and none of these $l$ vertices are connected to each other. Now, note that if $u_{a}^i = u_{x_i}^i$, then $u_{x_i}^i$ is connected to $v_{\overline{x}_i}$ and not to $v_{x_i}$. But $v_{x_i}$ is in $I$. We can conclude that the first $n$ vertices put into $I$ are not connected to any of the second collection of $l$ vertices added to $I$. Consequently, these vertices form an independent set of $n + l$ vertices.

   Conversely, suppose that $G$ has an independent set $I$ of size $n + l$. We describe a satisfying assignment $\sigma$ for $F$.

   $I$ can include at most one of $v_x$ and $v_{\overline{x}}$ for each $x$, as these vertices are connected. This provides up to $n$ vertices in $I$. Likewise, for each clause $C = a \lor b \lor c$, it can include at most one of $u_{a}^i, u_{b}^i, u_{c}^i$ as these three vertices are all connected (and similarly for a clause of two or one literals). This provides up to another $l$ vertices in $I$. As no other vertices are available, $I$ must include $n$ of the first group of vertices and $l$ of the second group.
Figure 2: The graph for the Independent Set construction for $F = (x_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (\overline{x}_2 \lor x_3)$. 
If \( v_x \in I \), then \( x \) is set to True in \( \sigma \), and if \( v_\overline{x} \in I \) then \( x \) is set to False in \( \sigma \). Next, we show that \( \sigma \) is a satisfying assignment for \( F \). For if \( u_i^x \in I \), if \( u_i^x = u_i^\overline{x} \), then \( v_x \in I \) (as \( u_i^\overline{x} \) is connected to \( v_x \)) and so \( x \) was set to True in \( \sigma \), causing \( C_i \) to evaluate to True; while if \( u_i^x = u_i^\overline{x} \), then \( v_x \in I \) (as \( u_i^\overline{x} \) is connected to \( v_x \)) and so \( x \) was set to False in \( \sigma \), again causing \( C_i \) to evaluate to True. As this holds for all \( l \) clauses, \( F \) evaluates to True when its Boolean assignments are given by \( \sigma \).

This shows the claim.

5. Clearly \( A_{3\text{-SAT}} \) runs in polynomial time if \( A_{\text{IS}} \) runs in polynomial time.

\[ \square \]

**Example 7** G-SAT.

**Input:** \( F \), a Boolean Formula.

**Question:** Does \( F \) have a satisfying assignment (of Boolean values to its variables), that is, an assignment that causes it to evaluate to True?

**Claim 8** Given a polynomial time algorithm for Satisfiability (SAT) there is a polynomial time algorithm for G-SAT.

**Proof:** Let \( F \) be the input to the G-SAT algorithm, \( A_{G\text{-SAT}} \). The algorithm proceeds as follows.

1. \( A_{G\text{-SAT}} \) computes \( E \), where \( E \) is a CNF formula, with the property that

\[
F \in \text{G-SAT} \iff E \in \text{SAT}.
\]

2. It runs \( A_{\text{SAT}}(E) \).

3. It reports the answer given by \( A_{\text{SAT}}(E) \).

\( A_{G\text{-SAT}} \) computes \( E \) in two steps. First, it rewrites \( F \) as the equivalent Boolean formula \( D \) in which all the “nots” are being applied directly to the literals. For example, if \( F = \neg((\neg(x_1 \lor \overline{x}_2) \land x_3) \lor (x_3 \land \overline{x}_2)) \), then \( D = [(x_1 \lor \overline{x}_2) \lor \overline{x}_3] \land (\overline{x}_3 \lor x_2) \).

\( D \) is computed by applying the rules \( \neg(x \lor y) = \overline{x} \land \overline{y} \) and \( \neg(w \land z) = \overline{w} \lor \overline{z} \), starting at the outermost level. As \( D \) always has the same truth value as \( F \), \( F \) is satisfiable exactly if \( D \) is satisfiable: \( F \in \text{G-SAT} \iff D \in \text{G-SAT} \).

Next \( A_{G\text{-SAT}} \) computes a CNF formula \( E \) such that \( E \) is satisfiable if and only if \( D \) is satisfiable. To help understand the process, let’s view \( D \) as an expression tree. For the above example, the corresponding expression tree is shown in Figure 3.

\( E \) receives a new variable for each internal node in the expression tree. Note that the leaves retain their literal labels (either \( x \) or \( \overline{x} \)). Let \( r \) be the variable
at the root. For a parent node with operator $\lor$ and corresponding variable $y$, with children having corresponding variables $z_1, z_2, \cdots, z_k$, several clauses are included in $E$, clauses that evaluate to True exactly if $y = z_1 \lor z_2 \lor \cdots \lor z_k$. The following clauses suffice:

$$B = (\bar{y} \lor z_1 \lor z_2 \lor \cdots \lor z_k) \land (y \lor \bar{z}_1) \land (y \lor \bar{z}_2) \land \cdots \land (y \lor \bar{z}_k).$$

For if $B$ evaluates to True (as is needed for $E$ to evaluate to True) then one of the following two situations applies:

- $y$ is set to True; then $\bar{y}$ is False and one of $z_1, z_2, \cdots, z_k$ must be set to True.
- $y$ is set to False; then all of $\bar{z}_1, \bar{z}_2, \cdots, \bar{z}_k$ are set to True, i.e. all of $z_1, z_2, \cdots, z_k$ are set to False.

Similarly, for a parent node with operator $\land$ and corresponding variable $y$, with children having corresponding variables $z_1, z_2, \cdots, z_k$, the following clauses are put into $E$:

$$B = (\bar{y} \lor z_1) \land (\bar{y} \lor z_2) \land \cdots \land (\bar{y} \lor z_k) \land (y \lor \bar{z}_1 \lor \bar{z}_2 \lor \cdots \lor \bar{z}_k).$$

For if $B$ evaluates to True then one of the following two situations applies:

- $y$ is set to True; then all of $z_1, z_2, \cdots, z_k$ must be set to True.
- $y$ is set to False; then at least one of $\bar{z}_1, \bar{z}_2, \cdots, \bar{z}_k$ is set to True, i.e. at least one of $z_1, z_2, \cdots, z_k$ is set to False.

So $E$ consists of the ‘and’ of these collections of clauses, one collection per internal node in the expression tree, ‘and’-ed with the clause $(r)$.

4. Now we show that $D \in \text{G-SAT} \iff E \in \text{SAT}$. 

Figure 3: Expression Tree Representation of $D = [(x_1 \lor \bar{x}_2) \lor \bar{x}_3] \land (\bar{x}_3 \lor x_2)$. 

First, suppose that $D$ is satisfiable. Let $\sigma$ be a satisfying assignment of Boolean values for $D$. Then the following Boolean assignment causes $E$ to evaluate to True: first, the same assignments are used in $E$ as in $D$ for their common variables. Next, the truth values under $\sigma$ are found for each node of the evaluation tree. For each internal node in $E$’s evaluation tree, the corresponding variable is set to the just determined truth value. As $D$ evaluates to True, $r$ is set to True, causing clause $(r)$ to evaluate to True. For each collection of clauses corresponding to an internal node of the tree, the relevant clauses state that the parent variable, under the given Boolean assignment, has the same truth value as the ‘or’ or ‘and’ of its children variables, as appropriate; thus each collection of clauses evaluates to True. Consequently $D$ evaluates to True under the just specified Boolean assignment.

Now suppose that $E$ is satisfiable. Let $\tau$ be a satisfying assignment of Boolean values for $E$. The same Boolean assignments are used in $D$ for their common variables. As each collection of clauses corresponding to an internal node evaluates to True, this tells us that the variables for the internal nodes take on the values the evaluation tree determines at its internal nodes. Thus $D$ evaluates to the Boolean value for $r$; but because clause $(r)$ in $E$ must evaluate to True, this means that $D$ evaluates to True under the given Boolean assignment.

This shows the claim.

5. Clearly $A_{G-SAT}$ runs in polynomial time if $A_{SAT}$ runs in polynomial time.

Claim 9 Given a polynomial time algorithm for 3-Satisfiability (3-SAT) there is a polynomial time algorithm for Satisfiability (SAT).

Proof:

Let $F$ be the input to the SAT algorithm, $A_{SAT}$. The algorithm proceeds as follows.

1. $A_{SAT}$ computes $E$, where $E$ is a CNF formula, with the property that

   $$F \in SAT \iff E \in 3\text{-SAT}.$$ 

2. It runs $A_{3\text{-SAT}}(E)$.

3. It reports the answer given by $A_{3\text{-SAT}}(E)$.

$A_{SAT}$ obtains $E$ as follows. First, any clause containing 3 or fewer literals is simply copied to $E$. Then for a longer clause $C = l_1 \lor l_2 \lor \cdots \lor l_k, k > 3$, it puts the following clauses into $E$:

$$D = (l_1 \lor l_2 \lor z_1) \wedge (z_1 \lor l_3 \lor z_2) \wedge \cdots \wedge (z_{k-4} \lor l_{k-2} \lor z_{k-3}) \wedge (z_{k-3} \lor l_{k-1} \lor l_k).$$
To see why this works it suffices to note that with the same Boolean assignments to the literals $l_1, l_2, \cdots, l_k$, $C$ and $D$ either can both evaluate to True, or neither can evaluate to True. For if $l_i$ is set to True, then setting $z_1, z_2, \cdots, z_{i-2}$ to True, and $z_{i-1}, z_i, \cdots, z_{k-3}$ to False causes $D$ to evaluate to True, while if all of $l_1, l_2, \cdots, l_k$ are set to False, then $D$ reduces to $(z_1) \land (z_1 \lor z_2) \land \cdots \land (z_{k-4} \lor z_{k-3}) \land (\overline{z}_{k-3})$, which always evaluates to False (for to make the first clause evaluate to True requires $z_1$ to be True, then to make the second clause evaluate to True requires $z_2$ to be True, and so on, causing all of $z_3, \cdots, z_{k-3}$ to be set to True; but then the final clause cannot evaluate to True).

4. Next we argue that $F \in \text{SAT} \iff E \in \text{3-SAT}$.

First, suppose that $F$ is satisfiable, using Boolean assignment $\sigma$. Then this assignment is applied to $E$ together with the extension to the new variables as described in the previous paragraph. Clearly this causes $E$ to evaluate to True also.

Next, suppose that $E$ is satisfiable, using Boolean assignment $\tau$. The common variables in $F$ are given the same assignment. Now suppose for a contradiction that some clause $C$ in $F$ evaluated to False. As argued above, the corresponding clauses $D$ in $E$ would evaluate to False; but $E$ evaluates to True and so every such $D$ must evaluate to True also. So in fact $C$ evaluates to True. As this holds for every clause in $F$, this means that $F$ evaluates to True also.

This shows the claim.

5. Clearly $A_{\text{SAT}}$ runs in polynomial time if $A_{3\text{-SAT}}$ runs in polynomial time. ■

We turn to a less obvious reduction.

Claim 10 Given a polynomial time algorithm for Directed Hamiltonian Circuit (DHC) there is a polynomial time algorithm for Vertex Cover (VC): VC $\leq_P$ DHC.

Proof: Let $(G, k)$ be the input to the VC algorithm, $A_{\text{VC}}$. The algorithm proceeds as follows.

1. $A_{\text{VC}}$ computes $H$, where $H$ is a directed graph, with the property that

   $$(G, k) \in \text{VC} \iff H \in \text{DHC}.$$  

2. It runs $A_{\text{DHC}}(H)$.

3. It reports the answer given by $A_{\text{DHC}}(H)$.

Let $G = (V, E)$. $A_{\text{VC}}$ obtains $H$ as follows. For each edge $e = (u, v) \in E$, it adds four vertices to $H$, namely $(u, e, 0), (u, e, 1), (v, e, 0), (v, e, 1)$, connected as
shown in Figure 3(a); this is called an \textit{edge gadget}. Then, for each vertex $u \in V$, with incident edges $e_1, e_2, \ldots, e_l$, the corresponding vertices, two per edge, are connected as shown in Figure 3(b). This can be thought of as corresponding to $u$’s adjacency list, except that there are two entries per edge; we call this sequence of $2k$ nodes the \textit{adjacency list} for $u$ (in $H$). Finally, $k$ \textit{selector} vertices $s_1, s_2, \ldots, s_k$ are added to $H$, together with edges from each $s_i$ to the first vertex on each vertex $u$’s “adjacency list” (i.e. edges $(s_i, (u, e_1, 0))$, and edges from the last vertex on each vertex $u$’s adjacency list to each $s_i$ (i.e. edges $((u, e_l, 1), s_i)$).

This completes the description of graph $H$. An example of a pair $(G, 2)$ and the corresponding $H$ are shown in Figure 3.

![Figure 4](image)

\textbf{Figure 4:} The structure in $H$ due to edge $(u, v)$ in $G$ and to $u$’s adjacency list for edges $e_1, e_2, \ldots, e_l$.

4. We argue that $H$ has a Hamiltonian Circuit exactly if $G$ has a vertex cover of size $k$.

First, suppose that $G$ has a vertex cover of size $k$, $U = \{u_1, u_2, \ldots, u_k\}$ say. A preliminary, and incomplete Hamiltonian Circuit $C'$ is given by the following cycle: It starts at $s_1$ and then goes through the adjacency list for vertex $u_1$, then goes to $s_2$, then traverses the adjacency list for $u_2$, and so on, eventually traversing the adjacency list for $u_k$, following which is completes the cycle by returning to $s_1$. $C'$ needs to be modified to include the nodes on the adjacency
Figure 5: The graph $H$ corresponding to $(G, 2)$. 
lists of vertices outside the vertex cover $U$. The nodes in $H$ which have not yet been traversed all correspond to the $v$ end of edges $(u, v)$ with just one endpoint, $u$, in $U$. (For as $U$ forms a vertex cover, there is no edge with zero endpoints in $U$.) The vertices not on $C'$ are $(v, e, 0)$ and $(v, e, 1)$. To include them, it suffices to replace the edge $((u, e, 0), (u, e, 1))$ with the sequence of three edges obtained by traversing $(u, e, 0), (v, e, 0), (v, e, 1)$ in that order. The result is still a cycle, but now it goes through every vertex of $H$ exactly once, so it forms a Hamiltonian Cycle.

Next, suppose that $H$ has a Hamiltonian Cycle $C$. Let $(u_i, e_{u_i,1}, 0)$ be the vertex following $s_i$ on $C$, for $1 \leq i \leq k$. Then we claim that $U = \{u_1, u_2, \ldots, u_k\}$ forms a vertex cover for $G$. To see this we need to understand how $C$ traverses the adjacency list of each of the $u_i$.

We claim that there are three ways for $C$ to traverse the edge gadget for edge $e = (u, v)$ in $G$, as illustrated in Figure 4. For there are two vertices by which the gadget can be entered ($(u, e, 0)$ and $(v, e, 0)$), and two by which it can be exited ($(v, e, 1)$ and $(u, e, 1)$). Either it is entered twice and exited twice, in which case the corresponding entry and exit vertices are joined by single edges (Figure 4(a)) or it is entered and exited once, in which case all four vertices lie on the path between the corresponding entry and exit vertices (Figure 4(b) or (c)).

Let $e_1, e_2, \ldots, e_l$ be the edges incident on some $u = u_i \in U$, and suppose the edges appear in that order on $u$’s adjacency list in $H$. As the first edge gadget on $u$’s adjacency list in $H$ is entered by edge $(s_i, (u, e_1, 0))$, $C$ must go directly or by zig-zag from $(u, e_1, 0)$ to $(u, e_1, 1)$, then directly to $(u, e_2, 0)$, then directly or by zig-zag from $(u, e_2, 0)$ to $(u, e_2, 1)$, then directly to $(u, e_3, 0)$, and so on until it reaches $(u, e_l, 1)$, from where it goes to some $s_{i'}, i' \neq i$. By renumbering indices if needed, we can allow $i' = i + 1 \mod k$.

Thus $C$ traverses $k$ adjacency lists directly; the other nodes in $H$ corresponding to edge endpoints are visited by means zig-zags. This shows that for each such
edge one endpoint is in \( U \). As all nodes in \( H \) are traversed, it follows that every endpoint in a non-traversed adjacency list must have its other endpoint in \( U \). Consequently \( U \) forms a vertex cover.

This shows the claim.

5. Clearly \( \mathcal{A}_{\text{VC}} \) runs in polynomial time if \( \mathcal{A}_{\text{DHC}} \) runs in polynomial time.

\[ \Box \]

**Claim 11** Given a polynomial time for General Satisfiability (G-SAT) there is a polynomial time algorithm for Directed Hamiltonian Circuit (DHC): \( \text{DHC} \leq \text{SAT} \).

**Proof:** Let \( G \) be the input to the DHC algorithm, \( \mathcal{A}_{\text{DHC}} \). The algorithm proceeds as follows.

1. \( \mathcal{A}_{\text{DHC}} \) computes \( F \), where \( F \) is a logical formula, with the property that

\[
G \in \text{DHC} \iff F \in \text{G-SAT}.
\]

2. It runs \( \mathcal{A}_{\text{G-SAT}}(F) \).

3. It reports the answer given by \( \mathcal{A}_{\text{G-SAT}}(F) \).

\( \mathcal{A}_{\text{DHC}} \) obtains \( F \) as follows. Let \( G = (V, E) \) and \( |V| = n \). \( F \) has \( E \cdot V \) variables \( y^i_{u,v} \), \( 1 \leq i \leq n \), and \((u, v) \in E\). The meaning of a variable is that \( y^i_{u,v} = \text{True} \) if and only if \((u, v)\) is the \( i \)th edge in the Hamiltonian Circuit (on choosing an arbitrary first edge). We write \( F \) as an “and” of subformulas which ensure that \( F \) can evaluate to True if and only if \( G \) has a Hamiltonian Circuit.

First, \( F^I_v \) ensures node \( v \) has at least one incoming edge.

\[
F^I_v = \bigvee_{u: (u, v) \in E} y^i_{u,v}, \quad 1 \leq i \leq n
\]

\( F^{I_2}_v \) ensures node \( v \) has at most one incoming edge.

\[
F^{I_2}_v = \bigwedge_{t \neq u} (\overline{y}^i_{u,v} \lor \overline{y}^j_{t,v}) \land \bigwedge_{u: (u, v) \in E} (\overline{y}^i_{u,v} \lor \overline{y}^j_{u,v}), \quad 1 \leq i < j \leq n
\]

Second, there are analogous constraints to ensure \( v \) has exactly one outgoing edge:
\[ F_{v}^{O_1} = \bigvee_{w: (v, w) \in E} y_{v,w}^{i}. \]

\[ F_{v}^{O_2} = \bigwedge_{w \neq x} (\overline{y}_{v,w}^i \lor \overline{y}_{v,x}^j) \land \bigwedge_{w: (v, w) \in E} (\overline{y}_{v,w}^i \lor \overline{y}_{v,w}^j). \]

The constraint \( \land_v (F_{v}^{I_1} \land F_{v}^{I_2} \land F_{v}^{O_1} \land F_{v}^{O_2}) \) if True ensures that \( G \) has one or more cycles that go through every vertex exactly once. To ensure that there is one cycle, subformula \( F^A \) is used to force the \( [(i + 1) \mod n] \)th edge to follow the \( i \)th edge, for each \( i, 1 \leq i \leq n \).

\[
F^A = \bigvee_{1 \leq i \leq n} (y_{uv}^i \Rightarrow \bigvee_{w: (v, w) \in E} y_{vw}^{(i+1) \mod n])
\]

Note that a formula \( C \Rightarrow D \) can be rewritten as \( \overline{C} \lor D \), so

\[
F^A = \bigvee_{1 \leq i \leq n} (\overline{y}_{uv}^i \lor \bigvee_{w: (v, w) \in E} y_{vw}^{(i+1) \mod n})
\]

Finally,

\[ F = F^A \land \bigwedge_v (F_{v}^{I_1} \land F_{v}^{I_2} \land F_{v}^{O_1} \land F_{v}^{O_2}). \]

4. We argue that \( G \in \text{DHC} \iff F \in \text{G-SAT} \).

First, suppose that \( F \in \text{G-SAT} \). Then if \( y_{uv}^i = \text{True} \), \( (u, v) \) is made the \( i \)th edge in the proposed Hamiltonian Circuit for \( G \). The truthfulness of \( F_{v}^{I_1} \) and of \( F_{v}^{I_2} \) ensures that exactly one edge into \( v \) is selected, and similarly \( F_{v}^{O_1} \) and \( F_{v}^{O_2} \) ensures that exactly one edge out of \( u \) is selected. Thus \( n = |V| \) edges must be selected. Finally \( F^A \) ensures that the \( i \)th and \( (i + 1) \)st edges (mod \( n \)) have a common endpoint, ensuring that the edges form a Hamiltonian Circuit.

Second, suppose that \( G \in \text{DHC} \). Let \( v_1, v_2, \ldots, v_n \) be a Hamiltonian Circuit for \( G \). We set \( y_{v,v_{i+1}}^i \) to True, and all other \( y \) variables to False. This causes \( F_{v}^{I_1} \), \( F_{v}^{I_2} \), \( F_{v}^{O_1} \), \( F_{v}^{O_2} \) to evaluate to True for each \( v \in V \), and causes \( F^A \) to evaluate to True also. So \( F \) evaluates to True.

This shows the claim.
Clearly $A_{HC}$ runs in polynomial time if $A_{G-SAT}$ runs in polynomial time.

## 2 NP-Completeness

We begin by showing that reductions compose.

**Lemma 12** If $J \leq_P K \leq_P L$ then $J \leq_P L$.

**Proof:** As $J \leq_P K$ there is a polynomial time computable function $f$ such that $x \in J \iff f(x) \in K$. And as $K \leq_P L$ there is a polynomial time computable function $g$ such that $y \in K \iff g(y) \in L$. So $x \in J \iff g(f(x)) \in L$, and $g \circ f$ is also polynomial time computable. That is, $J \leq_P L$.

**Definition 13** A language $L$ is NP-complete ($L \in NPC$) if

1. $L \in NP$, and
2. For every $J \in NP$, $J \leq_P L$.

**Lemma 14** If $K$ is NP-complete and we show that $L \in NP$, and $K \leq_P L$ then we can conclude $L$ is NP-Complete also.

**Proof:** It suffices to show that for all $J \in NP$, $J \leq_P L$. But as $K$ is NP-complete, $J \leq_P K$, and we know $K \leq_P L$. By Lemma 12, we conclude $J \leq_P L$.

Lemma 14 means that once we have shown one problem is NP-Complete, e.g. General Satisfiability, then subsequent problems can be shown NP-Complete by means of reductions from G-SAT, as already given in earlier sections.

**Example 15** Circuit Value (CV).

**Input:** $C$, a circuit comprising and, or, and not gates, with Boolean inputs $x_1, x_2, \cdots, x_n$ and a Boolean output $y$. (The circuit can be viewed as a directed graph, where the inputs are the names of vertices with no in-edges, the output is the name of a vertex with no out-edge, and every other vertex is labeled by one of the operators (and, or, or not); the latter vertices have one or two inedges as appropriate, but there are no limits on the number of outedges they could have.

**Question:** Is there an assignment of Boolean values to the inputs that causes the circuit output to evaluate to True?
Claim 16  Given a polynomial time algorithm for General Satisfiability (G-SAT) there is a polynomial time algorithm for Circuit Value (CV): CV ≤ G-SAT.

Proof: Let C be the input to the CV algorithm, A_CV. The algorithm proceeds as follows.

1. A_CV computes F, where F is a logical formula, with the property that
   \[ C \in CV \iff F \in G-SAT. \]

2. It runs A_{G-SAT}(F).

3. It reports the answer given by A_{G-SAT}(F).

   A_CV constructs F using the same method as in Claim 8. It first pushes all the not operations to the “leaves” of the circuit (the vertices with no inedges). Then it creates a new variable for each vertex of the circuit labeled by a logical operator, and for each such vertex it creates a collection of clauses that evaluate to True exactly if the inputs and output of the operator are as specified by the operator (e.g. for an and, the output is the and of the inputs). Finally, it also includes the clause \((y)\).

4. As argued in Claim 8, F can be satisfied exactly if C can be satisfied.

5. Clearly A_CV runs in polynomial time if A_{G-SAT} runs in polynomial time. 

   \[ \blacksquare \]

Theorem 17  (Cook’s Theorem, 1972). CV is NP Complete.

Proof: Ultimately, the proof amounts to viewing a polynomial time computation of a computer as nothing more than the evaluation of a giant, albeit polynomial sized, circuit. To this end, we view a computer as an ensemble of logic gates with Boolean (bit-level) data residing on the edges (or wires) of the circuit. Any basic operation (addition, comparison, etc.) is performed by a fairly modest piece of circuitry (of depth \(O(w)\) or maybe \(O(w^2)\), where \(w\) is the word length (64 bits on modern computers, and \(O(\log n)\) bits in the RAM model). Thus a single step of a computation can be represented by a circuit of size \(O(m \log m)\) where, \(m\), a polynomial in \(n\), the input size, is the amount of memory the computation is using. Now the output values from one step are simply the input values to the next step, which uses another copy of the same one-step circuit, for each step of the computation runs on the same computer.

The proof of this theorem will be showing how to solve problems that have a verification algorithm. The argument of the previous paragraph shows that a verification algorithm without its certificate, amounts to an instance of a circuit value problem,
with the unknown certificate corresponding to the unknown inputs to the instance of
the circuit value problem.

Let \( L \in \text{NP} \). We need to show that \( L \leq_{P} CV \). That is, given a polynomial time
algorithm for CV, we give a polynomial time algorithm for \( L \).

As \( L \in \text{NP} \), we know that \( L \) has a polynomial time verifier, \( V_L \) say. Suppose that
\( V_L \) runs in time bounded by a polynomial \( p \).

Let \( x \) be an input string to be tested for membership in \( L \).

1. \( \mathcal{A}_L \) builds the following polynomial sized circuit \( D(x) \) to compute the output
   of \( V_L \). The circuit will have \( p(x) \) levels. Each level simulates one step of the
   computer running \( V_L \). The input to the \( i \)th level of the circuit is just the memory
   contents of the computer at the start of its \( i \)th step (this includes the program
   counter, the pointer to the next program statement to be executed); the output
   of the \( i \)th step are the memory contents at the end of the \( i \)th step; this is also
   the input to the \((i + 1)\)th step.

   The output value \( \mathcal{A}_L \) seeks is the variable corresponding to the Boolean output
   of \( V_L \), at the \( p(x) \)th level. (One small detail is that if the algorithm terminates
   before \( p(x) \) steps, then in subsequent steps the configuration of the computer
   is unchanged so that the output variable has the same value after \( p(x) \) steps.)

   The question of whether there is a certificate for \( x \), i.e. whether \( x \in L \), amounts
to asking whether the circuit can have output True (the free variables in the
circuit are the inputs corresponding to the certificate).

2. \( \mathcal{A}_L \) runs \( \mathcal{A}_{CV}(D(x)) \).

3. It reports the answer given by \( \mathcal{A}_{CV}(D(x)) \).

4. We argue that \( x \in L \iff D(x) \in CV \).

   If \( x \in L \), then it has a polynomial size certificate \( c(x) \), and \( V(x, c(x)) = \text{"Recognize"} \). Then the circuit \( D(x) \), when its inputs are set to values rep-
   resenting \( c(x) \) will evaluate to True (corresponding to the “Recognize”).

   While if \( D(x) \) can evaluate to True with inputs set to \( \sigma \), then on setting the
   certificate \( c(x) \) to values corresponding to \( \sigma \), \( V(x, c(x)) \) evaluates to ‘Recognize.”

   This shows the claim.

5. As \( D(x) \) is of size polynomial in \( x \), \( \mathcal{A}_{CV}(D(x)) \) runs in polynomial time, and
   hence so does \( \mathcal{A}_L(x) \).

A more formal proof, as was given by Cook, could be based on a Turing Machine
based definition of polynomial time. Then a polynomial time verification algorithm
can be instantiated as a polynomial time Turing Machine. A corresponding circuit,
or in this case a G-SAT formula \( F \), is written, where \( F \) can evaluate to True exactly
if there is a certificate for the given input and verification algorithm. The details of
$F$ are similar to those of the formula in the proof of Claim 11.

**Corollary 18** GSAT, SAT, 3-SAT, VC, Clique, IS, DHC, UHC, DHP, UHP are all
NP-Complete.

**Proof:** We have shown all these languages are in NP. Cook’s Theorem, Lemma 12,
and the individual reductions already shown yield the claimed result.

**Definition 19** Language $L$ is NP-hard if for all $K \in \text{NP}$, $K \leq_P L$.

**Lemma 20** If $J$ is NP-hard and $J \leq_P L$, then $L$ is NP-hard.

**Proof:** This is proved in the same way as Lemma 14.