1 NP-completeness

1. Reducing Exact Cover to SAT

To show that 3-cover is poly-reducible to SAT we need to define a function $f$ that maps an instance $(X, C)$ of the Exact Cover problem to a propositional formula $\varphi$, such that $C$ contains an exact cover of $X$ iff $\varphi$ is satisfiable, and the function $f$ has to be computable in polynomial time.

Let $(X, C)$ be an instance of the Exact Cover problem, with $X = \{x_1, \ldots, x_n\}$ and $C = \{C_1, \ldots, C_m\}$. Then, we define:

\[ B_x = \{ i \mid x \in C_i \} \]

which is the set of indices of all sets in $C$ that contain the element $x$ in $X$. Constructing this set is in PTIME since for each of the $n$ elements in $X$ we need to check $m$ sets in $C$, each of constant size 3. Also define:

\[ I = \{ (i, j) \mid C_i \cap C_j \neq \emptyset \} \]

which is a set that contains all pairs $(i, j)$ of indices of intersecting sets in $C$. Clearly, constructing this auxiliary set $I$ is in PTIME since there are only $m(m-1)/2$ possible distinct pairs of sets in $C$, and each $C_i$ contains exactly 3 elements.

Once we have constructed these sets in polynomial time, we proceed to build the formula $\varphi$ which is $f(X, C)$. We define our set of propositions to be $\{c_i : 1 \leq i \leq m\}$. Intuitively, a satisfying assignment to $\varphi$ will assign $c_i$ to be 1 if $C_i$ is selected in the exact cover, and 0 otherwise. The required formula is

\[ \varphi = \bigwedge_{x \in X} \left( \bigvee_{i \in B_x} c_i \right) \bigwedge_{(i, j) \in I} (\neg c_i \lor \neg c_j) \]

It follows that the size and computation of this formula are both polynomial in the size of the Exact Cover instance since $|X| = n$ and $|B_x| \leq m$ for all $x$, and $|I| \leq m(m-1)/2$ as discussed above.

The purpose of the first half of $\varphi$ is to ensure that all of $X$ is covered, since at least one of the sets $C_i$ has to be included in $C'$ that contains each $x$. The purpose of the second half of $\varphi$ is to ensure that no two sets $C_i$ and $C_j$ that both contain the same $x$ are included in $C'$, since if both $C_i$ and $C_j$ contain $x$, $\neg(c_i \land c_j)$ ensures they are not included together. We show that this is a valid reduction.
Lemma. There exists \( C' \subseteq C \) such that \( C' \) is an exact cover of \( X \) iff \( \varphi \) is SAT

Proof:

- \( \rightarrow \) Let \( C' \) be an exact cover of \( X \). We define \( \tau = \{ c_i : C_i \in C' \} \). Consider \( (i,j) \in I \). Then, by definition of \( I \), \( C_i \cap C_j \neq \emptyset \). Since \( C' \) is an exact cover, every pair of sets in \( C' \) are disjoint, so we must have \( C_i \notin C' \) or \( C_j \notin C' \). Therefore, for all \( (i,j) \in I \), \( \tau = (-c_i \lor -c_j) \).

  Also, since \( C' \) is an exact cover of \( X \), \( \bigcup C' = X \). So, for all \( x \in X \), we have \( B_x \cap \{ i : C_i \in C' \} \neq \emptyset \). Therefore, for all \( x \in X \), \( \tau = \bigvee_{i \in B_x} c_i \). Thus \( \tau \models \varphi \) and \( \varphi \) is satisfiable.

- \( \leftarrow \) If \( \varphi \) is SAT then there exists a truth assignment \( \tau \) such that \( \tau \models \varphi \). Define \( C' = \{ C_i \in C : \tau(c_i) = 1 \} \). Consider \( C_i \) and \( C_j \) in \( C' \). Then \( \tau(c_i) = \tau(c_j) = 1 \), so \( \tau \models (-c_i \lor -c_j) \). Therefore \( (i,j) \notin I \) and \( C_i \cap C_j = \emptyset \).

  Let \( x \in X \). Since \( \tau \models \bigvee_{i \in B_x} c_i \), there exists \( i \in B_x \) such that \( \tau(c_i) = 1 \). So there exists \( C_i \in C' \) such that \( x \in C_i \). Since every \( x \) in \( X \) is contained in some element of \( C' \), and each element of \( C' \) is also a subset of \( X \), we have \( \bigcup C' = X \).

  So \( C' \) satisfies the two conditions for being an exact cover of \( X \).

This proves that Exact Cover is a polynomially reducible to SAT.

2. Reducing 3-SAT to Integer Programming

To reduce 3-SAT to IP, we need to define a function \( f \) that maps a 3-CNF formula \( \varphi \) to an instance of the Integer Programming (IP) problem, which is a set of linear inequalities, such that \( \varphi \) is satisfiable iff the set of inequalities has an integral solution. Further, \( f \) must be computable in polynomial time.

Let \( \varphi \) be a 3-CNF formula and let \( AP(\varphi) = \{ p_1, \ldots, p_n \} \) be the set of atomic propositions in \( \varphi \). We construct \( I = f(\varphi) \), the desired set of linear inequalities. First, we define our set of variables to be \( \{ x_i, y_i : 1 \leq i \leq n \} \) and we define a mapping \( v \) from the set of possible literals to the set of variables as follows:

\[
v(p_i) = x_i \text{ and } v(-p_i) = y_i
\]

Since logical literals can only have the values 1 (true) and 0 (false), and the variables in an IP problem are assumed to be integral, we add the following inequalities to \( I \) to ensure that each variable corresponding to a literal can only have a value of 0 or 1, and that variables assigned to complementary literals have opposite values: For all \( 1 \leq i \leq n \),

\[
\begin{align*}
x_i &\leq 0, \quad y_i \geq 0 \\
x_i + y_i &> 0 \\
x_i + y_i &\leq 1
\end{align*}
\]

These inequalities restrict each \( x_i \) and \( y_i \) to complementary boolean values. The computation and size of these inequalities is linear in the size of \( \varphi \) since there are 4 inequalities per atomic proposition. Next, for each clause \((\alpha \lor \beta \lor \gamma)\) in the 3-CNF formula \( \varphi \), we add the following inequality to \( I \):

\[
v(\alpha) + v(\beta) + v(\gamma) \geq 1
\]
The computation and size of $I$ is still linear in the length of $\varphi$ since we added one inequality per clause. Thus given $\varphi$, we can compute $f(\varphi) = I$ in polynomial time. We next show that $f$ is a valid reduction.

**Lemma.** $\varphi$ is SAT iff $I$ has an integral solution.

*Proof:*

- $\rightarrow$ If $\varphi$ is SAT then there exists a truth assignment $\tau$ such that $\tau \models \varphi$. Then, for all $1 \leq i \leq n$, assign $x_i = \tau(p_i)$ and $y_i = 1 - \tau(p_i)$. We claim that this assignment constitutes an integral solution to $I$. Clearly, $x_i \geq 0$ and $y_i \geq 0$ are satisfied. Similarly, $x_i + y_i = \tau(p_i) + 1 - \tau(p_i) = 1$, so $0 \leq x_i + y_i \leq 1$ is satisfied. Also, if $\tau \models p_i$, then $x_i = v(p_i) = 1$ and if $\tau \models \neg p_i$ then $y_i = v(\neg p_i) = 1 - \tau(p_i) = 1$. So for any literal $\alpha$, if $\tau \models \alpha$ then $v(\alpha) = 1$. Therefore, because $\tau$ satisfies each clause $(\alpha \lor \beta \lor \gamma)$ of $\varphi$, we have $v(\alpha) + v(\beta) + v(\gamma) \geq 1$. Thus the assignment satisfies each inequality in $I$ and so $I$ has an integral solution.

- $\leftarrow$ We first note that the inequalities $x_i \geq 0$, $y_i \geq 0$, $x_i + y_i \leq 1$ restrict the possible integral values of $x_i$ and $y_i$ to the set $\{0, 1\}$. Further, the inequality $x_i + y_i > 0$ ensures that $y_i = 1 - x_i$, that is $x_i$ and $y_i$ are complementary.

Given an integral solution to $I$, we define the truth assignment $\tau$ as $\tau(p_i) = x_i$, which is a valid truth assignment since all $x_i$ are only 0 or 1. Then $\tau \models p_i$ if $x_i = 1$ and $\tau \models \neg p_i$ if $y_i = 1 - x_i = 1$. So $\tau \models \alpha$ if $v(\alpha) = 1$ for each literal $\alpha$. Since the integral solution satisfies each inequality $v(\alpha) + v(\beta) + v(\gamma) \geq 1$, and each variable has value 0 or 1, we must have $v(\alpha) = 1$ for some literal $\alpha$ in each clause of $\varphi$. So $\tau \models \alpha$ for some literal $\alpha$ in each clause of $\varphi$. Therefore $\tau$ satisfies each clause of $\varphi$ and we have that $\varphi$ is satisfiable.

This proves that 3-SAT is polynomially reducible to IP.

3. NP and co-NP.

(a) $L \in \text{co-NP}$

\[\iff x \notin L \iff \exists y \in \Sigma^P(|x|), A(x, y) = 1\] (definition of co-NP)

\[\iff x \in \Sigma^* - L \iff \exists y \in \Sigma^P(|x|), A(x, y) = 1\] (since $x \notin L$ iff $x \in \Sigma^* - L$)

\[\iff \Sigma^* - L \in \text{NP}\] (definition of NP)

(b) First we need the following simple result:

**Lemma.** $L' \leq_p L$ iff $\Sigma^* - L' \leq_p \Sigma^* - L$.

*Proof:*

\[L' \leq_p L \iff \exists f \text{ s.t. } x \in L' \iff f(x) \in L\]

\[\iff \exists f \text{ s.t. } x \notin L' \iff f(x) \notin L\]

\[\iff \exists f \text{ s.t. } x \in \Sigma^* - L' \iff f(x) \in \Sigma^* - L\]

\[\iff \Sigma^* - L' \leq_p \Sigma^* - L \text{ (so the same } f \text{ is the reduction)}\]
Now to the main problem.

\[ L \in \text{co-NPC} \]

iff \( L \in \text{co-NP} \) and \( \forall L' \in \text{co-NP}, L' \leq_p L \) \hspace{2cm} \text{(definition of co-NPC)}

iff \( \Sigma^* - L \in \text{NP} \) and \( \forall L' \in \text{co-NP}, \Sigma^* - L' \leq_p \Sigma^* - L \) \hspace{2cm} \text{(from part (a) above)}

iff \( \Sigma^* - L \in \text{NP} \) and \( \forall (\Sigma^* - L') \in \text{NP}, \Sigma^* - L' \leq_p \Sigma^* - L \) \hspace{2cm} \text{(from lemma above)}

iff \( \Sigma^* - L \in \text{NP} \) and \( \forall L'' \in \text{NP}, L'' \leq_p \Sigma^* - L \) \hspace{2cm} \text{(from part (a) above)}

iff \( \Sigma^* - L \in \text{co-NPC} \) \hspace{2cm} \text{(definition of NPC)}

(c) Let \( L \) be co-NP complete and let \( L \) be in NP. Since \( L \) is in NP, there exist polynomial \( p \) and PTIME algorithm \( A \) such that for all \( x \in \Sigma^* \), \( x \in L \) iff there exists \( y \in \Sigma^{p(|x|)} \) with \( A(x, y) = 1 \).

Consider an arbitrary \( L' \in \text{co-NP} \). Then \( L' \leq_p L \). Let \( f \) be the required polytime reduction. Since \( f \) is polynomially computable, \( |f(x)| \) must be polynomial in \( |x| \). And because polynomials are closed under functional composition, there exists a polynomial \( p' \) such that \( p(|f(x)|) < p'(|x|) \) for all \( x \in \Sigma^* \). We can define an algorithm \( A' \) which given \( x \in \Sigma^* \) and a witness \( y \in \Sigma^{p'(|x|)} \), first computes \( f(x) \) and then calls \( A(f(x), y) \). So we have \( A'(x, y) = A(f(x), y) \). Since \( f \) and \( A \) are both polynomially computable, \( A' \) is also polynomially computable.

Then we have, for all \( x \in \Sigma^* \),

\[ x \in L' \]

iff \( f(x) \in L \) \hspace{2cm} \text{(because \( f \) is a reduction)}

iff \( \exists y \in \Sigma^{p(|f(x)|)}, A(f(x), y) = 1 \) \hspace{2cm} \text{(because \( L \) is in NP)}

iff \( \exists y \in \Sigma^{p'(|x|)}, A'(x, y) = 1 \) \hspace{2cm} \text{(because \( A'(x, y) = A(f(x), y) \))}

Thus \( L' \) is in NP. Since \( L' \) was arbitrary, this proves co-NP \( \subseteq \) NP.

Now let \( L'' \) be an arbitrary language in NP. Then \( \Sigma^* - L'' \) is in co-NP. Since we have proved that co-NP \( \subseteq \) NP, therefore \( \Sigma^* - L'' \) must also be in NP. But then \( \Sigma^* - (\Sigma^* - L'') = L'' \) is in co-NP. Thus NP \( \subseteq \) co-NP.

Therefore we have that co-NP = NP.
2 Satisfiability

1. Proof by induction on the structure of $\varphi$.

Basis: $\varphi$ is a atomic proposition. We have two subcases:

- $\varphi$ is $p$. Then $\varphi[p \mapsto \tau(p)] = \tau(p)$ and $\tau|_{AP(\varphi)-(p)} = \tau|_{\varnothing \cup \{p\}} = \varnothing$. Therefore, $\tau|_{AP(\varphi)-(p)} = \varphi[p \mapsto \tau(p)]$ if $\varnothing = \tau(p)$ if $\tau = \tau(p)$ if $\tau = 1$ if $\tau = p$.
- $\varphi$ is $q \neq p$. Then $\varphi[p \mapsto \tau(p)] = q$ and $AP(\varphi) = \{q\} \cup \{p\} = \{q\}$. By the relevance lemma, $\tau = q$ if $\tau|_{\{q\}} = q$.

Inductive Step: We have the following cases.

Case 1: $\varphi = \neg \theta$. In this case $AP(\varphi) = AP(\theta)$. Then, by the induction hypothesis, $\tau = \theta$ if $\tau|_{AP(\theta)-(p)} = \theta[p \mapsto \tau(p)]$. We have,

$$\tau = \varphi$$
$$\tau = (\neg \theta)$$
$$\tau \neq \theta$$
$$\tau = \theta[p \mapsto \tau(p)]$$

(by semantics of $\neg$)

(by IH)

(by semantics of $\neg$)

(by defn. of substitution)

(because $AP(\varphi) = AP(\theta)$)

Case 2: $\varphi = (\theta \circ \psi)$. Then by the induction hypothesis, we have that,

$$\theta(\tau) = (\theta[p \mapsto \tau(p)])(\tau|_{AP(\theta)-(p)})$$

(1)

$$\psi(\tau) = (\psi[p \mapsto \tau(p)])(\tau|_{AP(\psi)-(p)})$$

(2)

Then we have,

$$\tau = \varphi$$
$$\tau = (\theta \circ \psi)$$
$$\tau = \theta(\tau), \psi(\tau) = 1$$

(by correspondence between Philosopher’s and EE view)

(by IH, using equations 1 and 2 above)

(by the relevance lemma, since $AP(\theta) \cup AP(\psi) = AP(\varphi)$)

(by EE semantics)

(by definition of substitution)

(by correspondence between Philosopher’s and EE view)
2. We first prove that any formula in DNF can be converted into an equivalent formula in CNF and vice versa, using only the distributive laws for $\land$ and $\lor$. We prove this by a two step induction.

**Lemma 1:** If $\varphi$ is in CNF, and $q_i$ is a literal for $1 \leq i \leq n$, then $(\varphi \lor (q_1 \land \cdots \land q_n))$ can be converted to a formula in CNF using only the distributive laws.

**Proof:** Let $\varphi = \wedge_{i=1}^m c_i$ where $c_i$ is a disjunctive clause for $1 \leq i \leq m$.

**Basis:** $\varphi \lor q = (\wedge_{i=1}^m c_i) \lor q \equiv \wedge_{i=1}^m (c_i \lor q) = \theta$ (using the distributive laws). Now $\theta$ is in CNF, since $\theta$ is a disjunctive clause and $q$ is a literal, therefore $c_i \lor q$ is a disjunctive clause. Therefore $\theta$ is in CNF.

**Inductive Step:** Let $Q = \wedge_{i=1}^n q_i$. Then $\varphi \lor (q_1 \land \cdots \land q_{n+1}) = (\varphi \lor (Q \land q_{n+1}))$ $\equiv (\varphi \lor Q) \land (\varphi \lor q_{n+1})$ (using the distributive laws). Now by IH, $(\varphi \lor Q) \equiv \theta_1$ where $\theta_1$ is in CNF and using the basis, $(\varphi \lor q_{n+1}) \equiv \theta_2$ where $\theta_2$ is in CNF. Then $\theta_1 \land \theta_2$ is in CNF too. So $\varphi \lor (q_1 \land \cdots \land q_{n+1}) \equiv \theta_1 \land \theta_2 = \theta$ where $\theta$ is in CNF.

**Lemma 2:** If $\varphi$ is in DNF, then it can be converted to an equivalent formula in CNF using only the distributive laws.

**Proof:** If $\varphi$ is in DNF and contains a single clause, we can easily convert it to CNF by assigning each literal to its own singleton disjunctive clause. Let $\varphi = \lor_{i=1}^m c_i$ where $c_i$ is a conjunctive clause for $1 \leq i \leq m$ and $m > 1$. Let $\theta = \lor_{i=1}^{m-1} c_i$. Then $\varphi = (\theta \lor c_m)$. Then by IH, we have that $\theta \equiv \theta'$ where $\theta'$ is in CNF. Consider $\varphi' = (\theta' \lor c_m)$. Then $\varphi \equiv \varphi'$, and $\varphi'$ has the form required in Lemma 1 above. Therefore $\varphi' \equiv \varphi''$ where $\varphi''$ is in CNF. Thus $\varphi \equiv \varphi''$ where $\varphi''$ is in CNF.

We have shown that any formula in DNF can be converted to an equivalent formula in CNF. The reverse also holds true and the proof proceeds identically, because of the symmetry of the distributive laws with respect to $\land$ and $\lor$. Next we prove that the negation of a CNF formula is equivalent to some DNF formula using only De Morgan’s laws.

**Lemma 3:** If $\varphi$ is in CNF, then $(\neg \varphi)$ can be converted to an equivalent formula in DNF using only De Morgan’s laws.

**Proof:** We prove the statement by induction.

If $\varphi$ is in CNF and contains a single disjunctive clause, then $(\neg \varphi) = (\neg(l_1 \land l_2 \land \cdots \land l_k)) \equiv \neg l_1 \land \neg l_2 \land \cdots \land \neg l_k \equiv (l_1' \land l_2' \land \cdots \land l_k') = \varphi'$ by $k$ applications of De Morgan’s laws and replacing each negated literal $\neg l_i$ by its complementary literal $l_i'$. Then $\varphi'$ is in DNF.

Let $\varphi = \land_{i=1}^m c_i$, where $c_i$ is a disjunctive clause for $1 \leq i \leq m$ and $m > 1$. Let $\theta = \land_{i=1}^{m-1} c_i$. By IH, there exists a DNF formula $\theta'$ such that $\theta' \equiv \neg \theta$, and by the base case, there exists a DNF formula $\theta''$ such that $\theta'' \equiv \neg c_m$. Then $(\theta' \lor \theta'') \equiv ((\neg \theta) \lor (\neg c_m)) \equiv (\neg (\theta \land c_m)) = \varphi$ and $(\theta' \lor \theta'')$ is in DNF.

We now prove that any formula can be written in CNF using only de Morgan’s laws and the distributive laws. Proof by induction:

**Basis:** $\varphi = p$ for some proposition $p$. Then $\varphi$ is already in CNF.
3. The three sentences written by Boole can be interpreted in the following way. The so-called properties can be thought of as propositions that are true if the property is present, and false otherwise. Let us then define Prop = \{A, B, C, D, E\} the set of propositions in our world, where each proposition corresponds to each of the properties mentioned by Boole. Then we have:

\[
\neg A \land \neg C \rightarrow (E \land ((B \land \neg D) \lor (\neg B \land D))) \\
(A \land D \land \neg E) \rightarrow (B \leftrightarrow C) \\
(A \land (B \lor E)) \leftrightarrow ((C \land \neg D) \lor (\neg C \land D))
\]

Our interpretation of the first question is "what relationships exist among B, C and D if A=1?". By setting A=1 and using our solver (which works in CNF) to simplify
the formulas, we get as a result the following formulas containing B, C and D:

\begin{align*}
\neg B \lor C \lor D \\
\neg B \lor \neg D \\
B \lor D \\
B \lor C \lor D \\
C \lor \neg D \\
\neg C \lor D
\end{align*}

Note that the last two clauses are equivalent to \((C \land D) \lor (\neg C \land \neg D)\) which is equivalent to \(C \leftrightarrow D\), so when property A is present, properties C and D are either both present, or none. Likewise, from the second and third clauses, it can be inferred that \(B \leftrightarrow \neg D\), and from these two that \(B \leftrightarrow \neg C\) as well. The first and fourth equations above form a tautology, so they do not give any interesting information whatsoever. This concludes all analyses of the above clauses.

To look for "independent" relations among B, C, and D, our interpretation is that regardless of the value of A, if there is any relationship of the following that can be deduced from the first 3 original formulas:

\begin{align*}
B \rightarrow C \\
B \rightarrow D \\
C \rightarrow B \\
C \rightarrow D \\
D \rightarrow B \\
D \rightarrow C
\end{align*}

To check this, since we have proved before that \(\Phi \cup \neg \varphi\) is UNSAT iff \(\Phi \models \varphi\) (a la proof by contradiction), we can add the negation of the above relations to our list of clauses in CNF form and invoke the split solver; if it returns UNSAT, it means that the relation follows from the above. Unfortunately, plugging in all of the possible implications makes split return SAT, so this means that none of the above relations can be concluded from the premises.

For the second question, making \(B = 1\) yields the following clauses with only A, C and D:

\begin{align*}
C \lor D \\
A \lor \neg D \\
A \lor C \lor \neg D \\
A \lor \neg C \lor D \\
\neg A \lor C \lor D \\
\neg A \lor \neg C \lor \neg D
\end{align*}

In likewise manner, from the last two we can deduce \(A \leftrightarrow (C \lor D) \land (\neg C \lor \neg D)\), so if property A is present, one of C or D are also present, but not both. The third and
fourth imply that $\neg A \rightarrow (C \leftrightarrow D)$, so the absence of A makes C equal to D. Nothing interesting can be concluded from the first two though.

3 Adequacy

1. Inspecting the inequality we see that if $p_1 = 0$ then it cannot be satisfied, so $p_1$ must be 1. Once $p_1$ is 1, if either $p_2$ or $p_3$ are 1, then it is satisfied, but not if both are 0. So, a logically equivalent formula is:

$$\varphi_{ttot} = (p_1 \land (p_2 \lor p_3))$$

One way to prove $\varphi_{ttot} \models ttot(p_1, p_2, p_3)$ is to show that $\forall \tau \in 2^{Prop}$, where $Prop = \{p_1, p_2, p_3\}$, $ttot(\tau(p_1), \tau(p_2), \tau(p_3)) = \varphi_{ttot}(\tau)$. This is most easily done with a truth table, since $|2^{Prop}| = 8$.

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$ttot(\tau)$</th>
<th>$\varphi_{ttot}(\tau)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

2. $\text{succ-carry}(p, c, q)$. We assume that the $n^{th}$ bit is the least significant, so the increment propagates from the $n^{th}$ to the 1$^{st}$ position. We also assume the c vector must contain, for every $c_i$, the value of the carry produced by adding $c_{i+1}$ to $p_i$ (except for $c_n$ which contains the carry of adding 1 to $p_n$), so as a result $c_1 = 1$ iff there is overflow. The relation between $p, c, q$ is as follows:

$$p_n + 1 = q_n + 2c_n$$

and for all $1 \leq i < n$,

$$p_i + c_{i+1} = q_i + 2c_i$$

The relation between $p_n, c_n$ and $q_n$ is captured by the formula

$$\varphi_n = (p_n \land c_n \land \neg q_n) \lor (\neg p_n \land \neg c_n \land q_n)$$

Also, for any $1 \leq i < n$, the only possible assignments to $c_{i+1}$, $p_i$, $q_i$, and $c_i$ that ensure consistency with the increment operation are the following:

<table>
<thead>
<tr>
<th>$c_{i+1}$</th>
<th>$p_i$</th>
<th>$q_i$</th>
<th>$c_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
From this table, we can construct in DNF form the formula $\varphi_i$ that captures the relation between $c_i + 1, p_i, q_i, c_i$.

$$\varphi_i = \left[ \begin{array}{l} \neg c_i + 1 \land \neg p_i \land q_i \land c_i \\ c_i + 1 \land \neg p_i \land q_i \land \neg c_i \\ \neg c_i + 1 \land p_i \land q_i \land \neg c_i \\ c_i + 1 \land p_i \land \neg q_i \land c_i \end{array} \right] \lor$$

Then the following formula is logically equivalent to $\text{succ-carry}(p, c, q)$:

$$\varphi = \bigwedge_{1 \leq i \leq n} \varphi_i$$

By construction, this formula $\varphi$ ensures the correct increment/carry propagation so it only allows $(p, c, q)$ that represent the original number, bit-by-bit carry and result numbers respectively, and thus is logically equivalent to $\text{succ-carry}$. Also, since there is one constant-sized term for the last bit and there are four four-literal disjunctions for every bit from 1 to $n-1$, the size of $\varphi$ is $O(n)$.

The automaton is constructed as in the figure above. For simplicity of the figure, we provide an automaton that accepts any length of vectors $p, q, c$ (top) and then
we can intersect it with an automaton that accepts strings of length 3n only (bottom). The order in which the atomic propositions must be fed into this automaton is $p_n, q_n, c_n, \ldots, p_1, q_1, c_1$ (the figure explicitly shows which transition should correspond to which proposition between parenthesis next to the transition label - this is for clarification and is not part of the automaton of course). Also note that to avoid cluttering the figure, all unspecified transitions go to a ‘trap state’ that is never left, and so the automaton will not accept such strings.

The transitions for the automaton can be derived directly from the truth table above (the one used to build the formula) where only those combinations present in the table are the ones allowed. The states labeled with “last carry was 1” and “last carry was 0” help identify in which situation we are, i.e. if the addition of the last $p_i$ with $c_i+1$ produced a new carry $c_i$ or not. So, making “last carry was 1” the initial state is equivalent to adding 1 to $p$, and making both labeled states accepting ensures that the string seen so far corresponds to a valid increment of $p$.

The final automaton that accepts 3 vectors of length $n$ only is obtained by intersecting the top automaton with the bottom one, since the bottom one accepts all strings of length $3n$ and nothing else. Since the top automaton has 8 states (including the omitted ‘trap state’) and the bottom one has $3n+1$ states, the cartesian product automaton for the intersection has at most $24n+8$ states, which is $O(n)$.

3. $\{\lor, \land\}$ is not adequate.

Let $\tau_0 : Prop \to \{0,1\}$ be defined as $\tau_0(p) = 0$, for all $p \in Prop$. Then every function $f$ in $Form(\land, \lor)$ satisfies $f(\tau_0) = 0$. Proof by induction:

- Base Case: if $f = p \in Prop$ then $f(\tau_0) = p(\tau_0) = \tau_0(p) = 0$.
- Inductive Case: if $f = \lor(f_1, f_2)$. By I.H. $f_1(\tau_0) = 0$ and $f_2(\tau_0) = 0$, and then $f(\tau_0) = \lor(f_1(\tau_0), f_2(\tau_0)) = \lor(0,0) = 0$ by definition of $\lor$.
- $f = \land(f_1, f_2)$. By I.H. $f_1(\tau_0) = 0$ and $f_2(\tau_0) = 0$, and then $f(\tau_0) = \land(f_1(\tau_0), f_2(\tau_0)) = \land(0,0) = 0$ by definition of $\land$.

Now, the function $g = \neg p_k$ for some $p_k \in Prop$ does not satisfy the property since $g(\tau_0) = \neg(\tau_0(p_k)) = \neg(0) = 1$. Then, $g$ does not satisfy the property and thus is not in $Form(\lor, \land)$, so $g$ cannot be constructed with $\{\lor, \land\}$ and thus the set $\{\lor, \land\}$ is not adequate.

4. $\{\downarrow\}$ and $\{|\}$ are adequate.

|   |   | $p \downarrow q$ | $p|q$ |
|---|---|------------------|------|
| 0 | 0 | 1                | 1    |
| 0 | 1 | 0                | 1    |
| 1 | 0 | 0                | 1    |
| 1 | 1 | 0                | 0    |
5. \{false, \rightarrow\} is adequate. We prove by induction that for each formula \(\varphi \in Form^{(\neg, \lor)}\) there is a logically equivalent formula \(\psi \in Form^{(1)}\). Since we know that \{\neg, \lor\} is adequate, this suffices to prove that \{false, \rightarrow\} is adequate.

**Base Case**: \(\varphi\) is \(p\), where \(p \in \text{Prop}\). Then \(\varphi\) itself is in \(Form^{(1)}\).

**Inductive Step**: We have two subcases.

- \(\varphi = (\neg \theta)\). By IH, there exists \(\theta' \in Form^{(1)}\) such that \(\theta\) is logically equivalent to \(\theta'\). And, by the truth table for \(\neg\) above, we have \((\neg \theta') \models (\theta' \downarrow \theta')\).
  Therefore, \(\varphi \models (\theta' \downarrow \theta') \in Form^{(1)}\).

- \(\varphi = (\theta_1 \lor \theta_2)\). By IH, there exist \(\theta'_1, \theta'_2 \in Form^{(1)}\) such that \(\theta_1 \models \theta'_1\) and \(\theta_2 \models \theta'_2\), which implies \(\varphi \models (\theta'_1 \lor \theta'_2)\). And, by the truth table of \(\lor\), we have \((\theta'_1 \lor \theta'_2) \models ((\theta'_1 \downarrow \theta'_2) \downarrow ((\theta'_1 \downarrow \theta'_2) \downarrow (\theta'_1 \downarrow \theta'_2)) \in Form^{(1)}\).

**Base Case**: \(\varphi\) is \(p\), where \(p \in \text{Prop}\). Then \(\varphi\) itself is in \(Form^{(1)}\).

**Inductive Step**: We have two subcases.

- \(\varphi = (\neg \theta)\). By IH, there exists \(\theta' \in Form^{(1)}\) such that \(\theta\) is logically equivalent to \(\theta'\). And, by the truth table for \(|\) above, we have \((\neg \theta') \equiv (\theta' \neg \theta')\).
  Therefore, \(\varphi \equiv (\theta' \neg \theta') \in Form^{(1)}\).

- \(\varphi = (\theta_1 \land \theta_2)\). By IH, there exist \(\theta'_1, \theta'_2 \in Form^{(1)}\) such that \(\theta_1 \equiv \theta'_1\) and \(\theta_2 \equiv \theta'_2\), which implies \(\varphi \equiv (\theta'_1 \land \theta'_2)\). And, by the truth table of \(|\), we have \((\theta'_1 \land \theta'_2) \equiv ((\theta'_1 | \theta'_2) | (\theta'_1 | \theta'_2)) \in Form^{(1)}\).

---

5. \(\{false, \rightarrow\}\) is adequate.

We prove by induction that for each formula \(\varphi \in Form^{(\neg, \lor)}\) there is a logically equivalent formula \(\psi \in Form^{(\text{false, } \rightarrow)}\). Since we know that \(\{\neg, \lor\}\) is adequate, this suffices to prove that \(\{false, \rightarrow\}\) is adequate.

**Base Case**: \(\varphi\) is \(p\), where \(p \in \text{Prop}\). Then \(\varphi\) itself is in \(Form^{(\text{false, } \rightarrow)}\).

**Inductive Step**: We have two subcases.

- \(\varphi = (\neg \theta)\). By IH, there exists \(\theta' \in Form^{(\text{false, } \rightarrow)}\) such that \(\theta \equiv \theta'\). Also, \((\neg \theta) \equiv ((\neg \theta) \lor \text{false}) \equiv (\theta \rightarrow \text{false})\). Therefore, \(\varphi \equiv (\theta' \rightarrow \text{false}) \in Form^{(\text{false, } \rightarrow)}\).

- \(\varphi = (\theta_1 \lor \theta_2)\). By IH, there exist \(\theta'_1, \theta'_2 \in Form^{(\text{false, } \rightarrow)}\) such that \(\theta_1 \equiv \theta'_1\) and \(\theta_2 \equiv \theta'_2\), which implies \(\varphi \equiv (\theta'_1 \lor \theta'_2)\). Therefore, \(\varphi \equiv (\theta'_1 \lor \theta'_2) \equiv ((\theta'_1 | \theta'_2) | (\theta'_1 | \theta'_2)) \in Form^{(\text{false, } \rightarrow)}\).
6. $\text{+ is not adequate}$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p + q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Let $\tau_0 : \text{Prop} \rightarrow \{0, 1\}$ be defined as $\tau_0(p) = 0$, for all $p \in \text{Prop}$. Then every function $f$ in $\text{Form}^{(+)}$ satisfies $f(\tau_0) = 0$. Proof by induction:

- **Base Case:** if $f = p \in \text{Prop}$ then $f(\tau_0) = p(\tau_0) = \tau_0(p) = 0$.
- **Inductive Step:**
  - $f = +(f_1, f_2)$. By I.H. $f_1(\tau_0) = 0$ and $f_2(\tau_0) = 0$, and then $f(\tau_0) = +(f_1(\tau_0), f_2(\tau_0)) = +(0,0) = 0$ by definition of $\text{+}$.

Now, the function $g = \neg p_k$ for some $p_k \in \text{Prop}$ does not satisfy the property since $g(\tau_0) = \neg(\tau_0(p_k)) = \neg(0) = 1$. Then, $g$ does not satisfy the property and thus is not in $\text{Form}^{(\text{+})}$, so $g$ cannot be constructed with $\{\text{+}\}$ and thus the $\{\text{+}\}$ is not adequate.