

# COMP 409: Logic Homework 3

March 22, 2011

## 1 NP-completeness

### 1. *Exact cover*

To show that 3-cover is poly-reducible to SAT we need to find a mapping  $f : EC \rightarrow Form$  where  $(X, C) \in EC$  is an instance of an Exact Cover problem and  $\varphi \in Form$  is an instance of the SAT problem (and actually we will reduce it to 3-CNF), and the function  $f$  has to be computable in polynomial time.

Let us define  $X = \{x_1, \dots, x_n\}$  and  $C = \{C_1, \dots, C_m\}$  as stated in EC. Then, we define:

$$B_x = \{i \mid x \in C_i\}$$

which is the set of all set indices in  $C$  that contain the element  $x$  in  $X$ , and clearly 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 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 elements of  $C$ , and each  $C_i$  contains exactly 3 elements. Finally, let us define the propositions  $c_i$  to be 1 if  $C_i \in C'$  (i.e. if  $C_i$  is selected) and 0 otherwise.

Once we have constructed these sets in polynomial time, we proceed to build the formula  $\varphi$  which is  $f(X, C)$  defined in the following way:

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

Where it follows that the size and computation of this formula are both polynomial in the EC problem since  $|X| = n$  and  $|B_x| \leq m \forall x$ , 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  $x$ ,  $\neg(c_i \wedge c_j)$  ensures they are not included together.

Thus, if there is a satisfying truth assignment for  $\varphi$ , it corresponds to the values of the propositions  $c_i$  which in turn determine which sets to include in  $C'$ , which is the solution to the EC problem, and this is required to prove this is in fact a reduction.

**Lemma.**  $C' \subseteq C$  is an exact cover of  $X$  iff  $\varphi$  is SAT

*Proof:*

- $\rightarrow$  From the construction above, if  $C'$  is an exact cover of  $X$ , then the following hold: 1)  $C' \neq \emptyset$ , 2)  $C_i \cap C_j = \emptyset \forall i \neq j$  and  $C_i, C_j \in C'$  and 3)  $\cup C' = X$ . Then, taking the truth assignment  $\tau(c_i) = 1$  iff  $C_i \in C'$  it is clear that  $\tau \models \varphi$ .
- $\leftarrow$  If  $\varphi$  is SAT then there exists a truth assignment  $\tau$  s.t.  $\tau \models \varphi$ . Then the solution to the EC problem is to take  $C_i \in C'$  iff  $\tau(c_i) = 1$ , since  $\varphi$  is satisfiable if all of its clauses are true, which happens only if: 1) no  $x$  is included in more than one  $C_i \in C'$ , and 2) all  $x$  are accounted for in at least one  $C_i \in C'$  which follows from the construction of  $\varphi$ .

And we have proved that there is a polytime reduction from EC to SAT.

## 2. Integer Programming

To reduce 3-SAT to IP, we need to find a mapping  $f : 3 - SAT \rightarrow IP$  where  $\varphi$  in 3-CNF is an instance of 3-SAT and  $I$  is a set of linear inequalities and is an instance of IP, so that  $f(\varphi) = I$ . If  $AP(\varphi) = \{p_1, \dots, p_n\}$  is the set of atomic propositions in  $\varphi$ , then the *literals* in  $\varphi$  can be either  $p_i$  or  $\neg p_i$ , and we define:

$$v(p_i) = x_i \text{ and } v(\neg p_i) = y_i$$

to be the "variables" we will use in the inequalities in  $I$ . Since logical literals can only have the values 1 (true) and 0 (false), and the variables in an IP program are assumed to be integral, adding the following inequalities to the set  $I$  ensures that each literal has one of the values 0 or 1, and that the complementary literal has the opposite value:

$$x_i \geq 0, y_i \geq 0$$

$$x_i + y_i > 0$$

$$x_i + y_i \leq 1$$

For all  $1 \leq i \leq n$ , since imposing that  $x_i + y_i = 1$  means "exclusive or" and has the desired effect (note that the computation and size of these inequalities is linear on the size of  $\varphi$  since there are 4 inequalities per atomic proposition). Now that we have ensured that these are valid

literals, we add to the set  $I$ , for each clause  $(\alpha \vee \beta \vee \gamma)$  in the 3-CNF formula  $\varphi$ , the following inequality:

$$v(\alpha) + v(\beta) + v(\gamma) \geq 1$$

This inequality ensures that the clause it was derived from is satisfied, since the variables  $v(\alpha)$ ,  $v(\beta)$  and  $v(\gamma)$  that satisfy this inequality will make at least one of the literals  $\alpha$ ,  $\beta$  or  $\gamma$  true for all clauses, which satisfies  $\varphi$ . Again, note that the computation and size of  $I$  is still linear on the length of  $\varphi$  since there is one inequality per clause. Now that we have constructed the set of IP inequalities  $I$  from the 3-SAT formula  $\varphi$  (which defines the mapping  $f$ ) we can show it is a reduction.

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

*Proof:*

- $\rightarrow$  If  $\varphi$  is SAT then there exists a truth assignment  $\tau$  s.t.  $\tau \models \varphi$ . Then, take  $x_i = 1$  iff  $\tau(p_i) = 1$  and  $y_i = 1$  iff  $\tau(p_i) = 0$ . This assignment constitutes an integral solution to  $I$  because since  $\tau \models \varphi$ , all clauses in  $\varphi$  are true and thus all inequalities  $v(\alpha) + v(\beta) + v(\gamma) \geq 1$  are satisfied because at least one of the literals  $\alpha$ ,  $\beta$  or  $\gamma$  make one of the variables  $v(\alpha)$ ,  $v(\beta)$  or  $v(\gamma)$  equal to 1. From our definition of "variables"  $v()$ , the first inequalities are also satisfied since only one of  $p_i$  or  $\neg p_i$  is true.
- $\leftarrow$  If  $I$  has an integral solution, then take  $\tau(p_i) = x_i$ , which is a valid truth assignment since all  $x_i$  are only 0 or 1. This truth assignment satisfies  $\varphi$  because the variables  $v(\alpha)$ ,  $v(\beta)$  and  $v(\gamma)$  satisfy all inequalities, so at least one of them is 1, which means one of the literals  $\alpha$ ,  $\beta$  or  $\gamma$  is true, and thus all clauses  $(\alpha \vee \beta \vee \gamma)$  in  $\varphi$  are true, so  $\varphi$  is satisfiable.

And we have proved that there is a polytime reduction from 3-SAT to IP.

### 3. NP and co-NP.

- (a)  $L \in \text{co-NP}$  iff  $x \notin L$  iff  $\exists y \in \Sigma^{\leq P(|x|)}$  s.t.  $A(x, y) = 1$   
iff  $x \in \Sigma^* - L$  iff  $\exists y \in \Sigma^{\leq P(|x|)}$  s.t.  $A(x, y) = 1$   
iff  $\Sigma^* - L$  is in NP.  
(since  $x \notin L$  iff  $x \in \Sigma^* - L$ )

- (b) First we need the following simple result:

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

*Proof:*

$$\begin{aligned} L' \leq_p L & \text{ iff } \exists f \text{ s.t. } x \in L' \text{ iff } f(x) \in L \\ & \text{ iff } \exists f \text{ s.t. } x \notin L' \text{ iff } f(x) \notin L \\ & \text{ iff } \exists f \text{ s.t. } x \in \Sigma^* - L' \text{ iff } f(x) \in \Sigma^* - L \\ & \text{ iff } \Sigma^* - L' \leq_p \Sigma^* - L \text{ (so the same } f \text{ is the reduction)} \end{aligned}$$

Now to the main problem.

$L \in \text{co-NPC}$  iff  $L \in \text{co-NP}$  and  $L' \leq_p L \forall L' \in \text{co-NP}$  (by def.)

From part a)  $L \in \text{co-NP}$  iff  $\Sigma^* - L \in \text{NP}$ , and from the lemma  $L' \leq_p L$  iff  $\Sigma^* - L' \leq_p \Sigma^* - L$ . Substituting in the co-NP definition above we get:

$L \in \text{co-NPC}$  iff  $\Sigma^* - L \in \text{NP}$  and  $\Sigma^* - L' \leq_p \Sigma^* - L \forall L' \in \text{co-NP}$ .

Now, the quantifier  $\forall L' \in \text{co-NP}$  is equivalent to  $\forall L' \text{ s.t. } \bar{L}' \in \text{NP}$  (where  $\bar{L}'$  denotes the complement of  $L'$ ), and since every language has a unique complement this is also equivalent to  $\forall \bar{L}' \text{ s.t. } L' \in \text{co-NP}$  which is equivalent to  $\forall \bar{L}' \in \text{NP}$ . So, substituting this equivalent quantifier in the above definition and replacing  $\Sigma^* - L'$  with  $\bar{L}'$ :

$$\begin{aligned} L \in \text{co-NPC} & \quad \text{iff} \quad \Sigma^* - L \in \text{NP} \text{ and } \bar{L}' \leq_p \Sigma^* - L \forall \bar{L}' \in \text{NP}. \\ & \quad \text{iff} \quad \Sigma^* - L \text{ is NPC.} \end{aligned}$$

Which completes the proof.

- (c) Let  $L$  be co-NP complete and let  $L$  be in NP. Then by result (a) above,  $\Sigma^* - L$  is in co-NP, and by result (b),  $\Sigma^* - L$  is NP-complete. Since  $L$  is co-NP complete, therefore  $\Sigma^* - L \leq_p L$ . This means that  $L$  is in NP and an NP-complete problem can be polynomially reduced to  $L$ . Thus  $L$  is NP-complete.

Now consider  $A \in \text{NP}$ . Then  $A \leq_p L$  since  $L$  is NP-complete. But  $L$  is also in co-NP. Thus  $A$  can be polynomially reduced to a problem in co-NP. This implies  $A$  is in co-NP itself. Thus  $\text{NP} \subseteq \text{co-NP}$ .

Similarly, if we consider  $B \in \text{Co-NP}$ . Then  $A \leq_p L$  since  $L$  is co-NP complete. But  $L$  is also in NP. Thus  $B$  can be polynomially reduced to a problem in NP. This implies  $B$  is in NP itself. Thus  $\text{co-NP} \subseteq \text{NP}$ . Therefore we have that  $\text{co-NP} = \text{NP}$ .

## 2 Satisfiability

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

**Basis:** If  $\varphi$  is  $p$  then  $\tau \models \varphi$  iff  $\tau(p) = 1$ . In that case,  $\varphi[p \mapsto \tau(p)] = p[p \mapsto \tau(p)] = \tau(p)$  which is valid and so  $\tau|_{AP(\varphi) - \{p\}} \models \varphi[p \mapsto \tau(p)]$ . If  $\varphi$  is  $q \neq p$ , then  $\tau \models \varphi$  iff  $\tau|_{\{q\}} \models \varphi$  (by the relevance lemma) and  $\varphi[p \mapsto \tau(p)] = \varphi$  and  $AP(\varphi) - \{p\} = \{q\} - \{p\} = \{q\}$ . Therefore  $\tau \models \varphi$  iff  $\tau|_{AP(\varphi) - \{p\}} \models \varphi[p \mapsto \tau(p)]$ .

**Inductive Step:** We have the following cases.

**Case 1:**  $\varphi = (\neg\theta)$ . In this case  $AP(\varphi) = AP(\theta)$ . Then by the induction hypothesis,  $\tau \models \theta$  iff  $\tau|_{AP(\theta)-\{p\}} \models \theta[p \mapsto \tau(p)]$ . Then  $\tau \not\models \theta$  iff  $\tau|_{AP(\theta)-\{p\}} \not\models \theta[p \mapsto \tau(p)]$  and so,  $\tau(\theta) = 0$  iff  $\tau|_{AP(\theta)-\{p\}}(\theta[p \mapsto \tau(p)]) = 0$ . We use this in the proof below.

$$\begin{aligned}
& \tau \models \varphi \\
\text{iff } & \tau(\neg\theta) = 1 \\
\text{iff } & \tau(\theta) = 0 \\
\text{iff } & \tau|_{AP(\theta)-\{p\}}(\theta[p \mapsto \tau(p)]) = 0 \quad (\text{by IH}) \\
\text{iff } & \tau|_{AP(\theta)-\{p\}}(\neg\theta[p \mapsto \tau(p)]) = 1 \\
\text{iff } & \tau|_{AP(\varphi)-\{p\}}(\varphi[p \mapsto \tau(p)]) = 1 \\
\text{iff } & \tau|_{AP(\varphi)-\{p\}} \models \varphi[p \mapsto \tau(p)]
\end{aligned}$$

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

$$\begin{aligned}
\tau(\theta) &= \tau|_{AP(\theta)-\{p\}}(\theta[p \mapsto \tau(p)]) \\
\tau(\psi) &= \tau|_{AP(\psi)-\{p\}}(\psi[p \mapsto \tau(p)])
\end{aligned}$$

Therefore

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

Then

$$\begin{aligned}
& \tau \models \varphi \\
\text{iff } & \tau(\theta \circ \psi) = 1 \\
\text{iff } & \circ(\tau(\theta), \tau(\psi)) = 1 \\
\text{iff } & \circ(\tau|_{AP(\theta)-\{p\}}(\theta[p \mapsto \tau(p)]), \tau|_{AP(\psi)-\{p\}}(\psi[p \mapsto \tau(p)])) = 1 \quad (\text{by IH}) \\
\text{iff } & \tau|_{AP(\theta) \cup AP(\psi) - \{p\}}(\theta[p \mapsto \tau(p)] \circ \psi[p \mapsto \tau(p)]) = 1 \\
\text{iff } & \tau|_{AP(\varphi)-\{p\}}((\theta \circ \psi)[p \mapsto \tau(p)]) = 1 \quad (\text{since } AP(\theta) \cup AP(\psi) = AP(\varphi)) \\
\text{iff } & \tau|_{AP(\varphi)-\{p\}}(\varphi[p \mapsto \tau(p)]) = 1 \\
\text{iff } & \tau|_{AP(\varphi)-\{p\}} \models \varphi[p \mapsto \tau(p)]
\end{aligned}$$

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  $\wedge$  and  $\vee$ . 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  $(\varphi \vee (q_1 \wedge \dots \wedge q_n))$  can be converted to a formula in CNF using only the distributive laws.

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

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

**Inductive Step:** Let  $Q = \bigwedge_{i=1}^n q_i$ . Then  $\varphi \vee (q_1 \wedge \cdots \wedge q_{n+1}) = (\varphi \vee (Q \wedge q_{n+1})) \models ((\varphi \vee Q) \wedge (\varphi \vee q_{n+1}))$  (using the distributive laws). Now by IH,  $(\varphi \vee Q) \models \theta_1$  where  $\theta_1$  is in CNF and using the basis,  $(\varphi \vee q_{n+1}) \models \theta_2$  where  $\theta_2$  is in CNF. Then  $\theta_1 \wedge \theta_2$  is in CNF too. So  $\varphi \vee (q_1 \wedge \cdots \wedge q_{n+1}) \models \theta_1 \wedge \theta_2 = \theta$  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  $\varphi = \bigvee_{i=1}^m c_i$  where  $c_i$  is a conjunctive clause for  $1 \leq i \leq m$  and  $m > 1$ . Let  $\theta = \bigvee_{i=1}^{m-1} c_i$ . Then  $\varphi = (\theta \vee c_m)$ . Then by IH, we have that  $\theta \models \theta'$  where  $\theta'$  is in CNF. Consider  $\varphi' = (\theta' \vee c_m)$ . Then  $\varphi \models \varphi'$ , and  $\varphi'$  has the form required in Lemma 1 above. Therefore  $\varphi' \models \varphi''$  where  $\varphi''$  is in CNF. Thus  $\varphi \models \varphi''$  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  $\wedge$  and  $\vee$ .

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  $\varphi \models ((p \vee p) \wedge (p \vee p))$ .

**Inductive step:** We have the following cases:

Case 1:  $\varphi = \neg\theta$ . Then by IH  $\theta \models \theta'$  where  $\theta'$  is in CNF. Let  $\theta' = \bigwedge_{i=1}^m c_i$  where each  $c_i$  is a disjunctive clause. Then  $\neg\theta' = \neg(\bigwedge_{i=1}^m c_i) \models \bigvee_{i=1}^m (\neg c_i)$  (using de Morgan's laws). Further,  $\neg c_i \models |c'_i$  where each  $c'_i$  is a *conjunctive clause*. Thus  $\varphi' = \bigvee_{i=1}^m c'_i$  is a formula in DNF, and  $\varphi \models \varphi'$ . Now by the lemma above, we can obtain a formula  $\varphi''$  in CNF that is equivalent to  $\varphi'$  (since  $\varphi'$  is in DNF). Thus  $\varphi \models \varphi''$  where  $\varphi''$  is a formula in CNF.

Case 2:  $\varphi = \theta \wedge \psi$ . Then by IH  $\theta \models \theta'$  and  $\psi \models \psi'$ , where  $\theta'$  and  $\psi'$  are in CNF. Then  $\varphi' = \theta' \wedge \psi'$  is in CNF too and  $\varphi \models \varphi'$ .

Case 3:  $\varphi = \theta \vee \psi$ . Then by IH  $\theta \models \theta'$  and  $\psi \models \psi'$ , where  $\theta'$  and  $\psi'$  are in CNF. Further, by the lemma above, there exist DNF formulas,  $\theta''$  and  $\psi''$  such that  $\theta' \models \theta''$  and  $\psi' \models \psi''$ . Then  $\theta'' \vee \psi''$  is in DNF too and again by the lemmas above, we can find a formula  $\varphi''$  in CNF such that  $\varphi'' \models \theta'' \vee \psi''$ . Since  $\theta'' \vee \psi'' \models \theta' \vee \psi' \models \theta \vee \psi = \varphi$ , therefore  $\varphi \models \varphi''$  where  $\varphi''$  is a formula in CNF.

Case 4:  $\varphi = \theta \rightarrow \psi$ . Then  $\varphi \models \neg\theta \vee \psi$ . The size of  $\neg\theta$  is smaller than the size of  $\theta \rightarrow \psi$ , so we can apply the induction hypothesis to

it. By IH, there exist formulas  $\theta'$  and  $\psi'$  in CNF such that  $\neg\theta \models \theta'$  and  $\psi \models \psi'$ . Then by the lemmas above, we can find formulas  $\theta''$  and  $\psi''$  in DNF such that  $\theta'' \models \theta'$  and  $\psi'' \models \psi'$ . Then  $\theta'' \vee \psi''$  is in DNF too and we can apply the lemma again to obtain a formula  $\varphi'$  in CNF such that  $\varphi' \models \theta'' \vee \psi''$ . Also we have,  $\theta'' \vee \psi'' \models \theta' \vee \psi' \models \neg\theta \vee \psi = \varphi$ . Therefore we have  $\varphi \models \varphi'$  where  $\varphi'$  is in CNF.

Case 5:  $\varphi = \theta \leftrightarrow \psi$ . Then  $\varphi \models (\theta \rightarrow \psi) \wedge (\psi \rightarrow \theta)$ . Using case 4 above, we can find formulas  $\theta'$  and  $\psi'$  in CNF such that  $\theta' \models (\theta \rightarrow \psi)$  and  $(\psi \rightarrow \theta)$ . Then  $\theta' \wedge \psi'$  is in CNF too and also  $\theta' \wedge \psi' \models (\theta \rightarrow \psi) \wedge (\psi \rightarrow \theta) = \varphi$ . Therefore  $\varphi$  is equivalent to a formula in CNF.

Assume that there exists a poly-time algorithm that translates each formula to its conjunctive normal form. We know that VALID is co-NP complete for formulas in DNF. However, it is also known that VALID is in P for formulas in CNF. Thus if we have a poly-time algorithm for translating formulas into equivalent formulas in CNF, given a formula in DNF we could find its equivalent formula in CNF in poly-time and then check the equivalent formula for validity, again in poly-time. Thus we would be able to solve VALID for DNF in poly-time. This would imply that co-NP is a subset of P (since VALID for DNF is co-NP complete). Since, P is contained in co-NP (in fact in the intersection of NP and co-NP), we would have that co-NP = P. Since P is closed under complementation, we would have  $p = \text{co-P} = \text{co}(\text{co-NP}) = \text{NP}$ . Thus the existence of such an algorithm would lead to the surprising result that  $P = NP$ .

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:

$$(\neg A \wedge \neg C) \rightarrow (E \wedge ((B \wedge \neg D) \vee (\neg B \wedge D))) \quad (1)$$

$$(A \wedge D \wedge \neg E) \rightarrow (B \leftrightarrow C) \quad (2)$$

$$(A \wedge (B \vee E)) \leftrightarrow ((C \wedge \neg D) \vee (\neg C \wedge D)) \quad (3)$$

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:

$$\begin{aligned} &\neg B \vee C \vee D \\ &\neg B \vee \neg D \\ &B \vee D \\ &B \vee C \vee D \\ &C \vee \neg D \\ &\neg C \vee D \end{aligned}$$

Note that the last two clauses are equivalent to  $(C \wedge D) \vee (\neg C \wedge \neg D)$  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:

$$\begin{aligned} &B \rightarrow C \\ &B \rightarrow D \\ &C \rightarrow B \\ &C \rightarrow D \\ &D \rightarrow B \\ &D \rightarrow C \end{aligned}$$

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:

$$\begin{aligned}
& C \vee D \\
& A \vee \neg D \\
& A \vee C \vee \neg D \\
& A \vee \neg C \vee D \\
& \neg A \vee C \vee D \\
& \neg A \vee \neg C \vee \neg D
\end{aligned}$$

In likewise manner, from the last two we can deduce  $A \leftrightarrow (C \vee D) \wedge (\neg C \vee \neg D)$ , 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:

$$\varphi_{ttof} = (p_1 \wedge (p_2 \vee p_3))$$

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

| $p_1$ | $p_2$ | $p_3$ | $ttof(\tau)$ | $\varphi_{ttof}(\tau)$ |
|-------|-------|-------|--------------|------------------------|
| 0     | 0     | 0     | 0            | 0                      |
| 0     | 0     | 1     | 0            | 0                      |
| 0     | 1     | 0     | 0            | 0                      |
| 0     | 1     | 1     | 0            | 0                      |
| 1     | 0     | 0     | 0            | 0                      |
| 1     | 0     | 1     | 1            | 1                      |
| 1     | 1     | 0     | 1            | 1                      |
| 1     | 1     | 1     | 1            | 1                      |

2. *succ-carry*( $\mathbf{p}, \mathbf{c}, \mathbf{q}$ ). 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  $\mathbf{c}$  vector is to 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 following formula is logically equivalent to *succ-carry*( $\mathbf{p}, \mathbf{c}, \mathbf{q}$ ):

$$\varphi = (p_n \wedge c_n \wedge \neg q_n) \vee (\neg p_n \wedge \neg c_n \wedge q_n) \dots$$

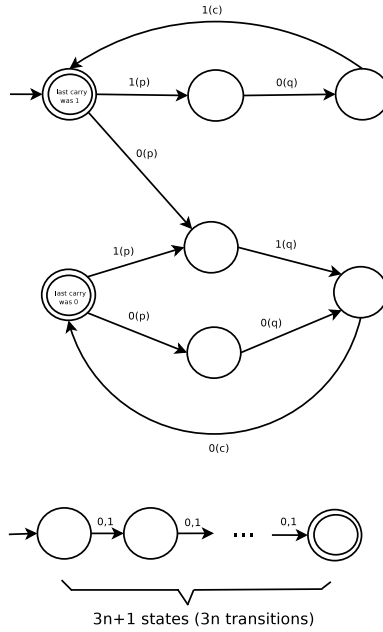
Then, for any  $1 \leq i \leq n-1$ , the following are the only possible assignments that ensure consistency with the increment operation:

| $c_{i+1}$ | $p_i$ | $q_i$ | $c_i$ |
|-----------|-------|-------|-------|
| 0         | 0     | 0     | 0     |
| 1         | 0     | 1     | 0     |
| 0         | 1     | 1     | 0     |
| 1         | 1     | 0     | 1     |

From this table, we can construct in DNF form the rest of  $\varphi$  by adding conjunctions that make true each of the combinations in the above table:

$$\cdots \bigwedge_{1 \leq i \leq n-1} \left[ \begin{array}{l} ( \neg c_{i+1} \wedge \neg p_i \wedge q_i \wedge c_i ) \vee \\ ( c_{i+1} \wedge \neg p_i \wedge q_i \wedge \neg c_i ) \vee \\ ( \neg c_{i+1} \wedge p_i \wedge q_i \wedge \neg c_i ) \vee \\ ( c_{i+1} \wedge p_i \wedge \neg q_i \wedge c_i ) \end{array} \right]$$

By construction, this formula  $\varphi$  ensures the correct increment/carry propagation so it only allows  $(\mathbf{p}, \mathbf{c}, \mathbf{q})$  that represent the original number, bit-by-bit carry and result numbers respectively, and thus is logically equivalent to *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  $\mathbf{p}, \mathbf{q}, \mathbf{c}$

(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  $p_n, q_n, c_n, \dots, p_1, q_1, c_1$  (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 cross product automaton for the intersection has at most  $24n + 8$  states, which is  $O(n)$ .

3.  $\{\vee, \wedge\}$  is not adequate.

Every function  $f$  in  $Form^{\{\wedge, \vee\}}$  satisfies  $f(\tau_0) = 0$  where  $\tau_0(p_i) = 0 \forall p_i \in Prop$ . Proof by induction:

- Base: if  $f = p_i \in Prop$  then  $f(\tau_0) = \tau_0(f) = \tau_0(p_i) = 0$ .
- Ind:
  - $f = \vee(f_1, f_2)$ . By I.H.  $f_1(\tau_0) = 0$  and  $f_2(\tau_0) = 0$ , and then  $f(\tau_0) = \vee(f_1(\tau_0), f_2(\tau_0)) = \vee(0, 0) = 0$  by definition of  $\vee$ .
  - $f = \wedge(f_1, f_2)$ . By I.H.  $f_1(\tau_0) = 0$  and  $f_2(\tau_0) = 0$ , and then  $f(\tau_0) = \wedge(f_1(\tau_0), f_2(\tau_0)) = \wedge(0, 0) = 0$  by definition of  $\wedge$ .

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  $Form^{\{\vee, \wedge\}}$ , so  $g$  cannot be constructed with  $\{\vee, \wedge\}$  and thus the set  $\{\vee, \wedge\}$  is not adequate.

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

| $p$ | $q$ | $p \downarrow q$ | $p q$ |
|-----|-----|------------------|-------|
| 0   | 0   | 1                | 1     |
| 0   | 1   | 0                | 1     |
| 1   | 0   | 0                | 1     |
| 1   | 1   | 0                | 0     |

- $\{\downarrow\}$  is adequate. We know that the set  $\{\neg, \vee\}$  is adequate, so all boolean functions can be implemented with  $\{\neg, \vee\}$ . If we can prove that we can translate any formula  $\varphi \in Form^{\{\neg, \vee\}}$  to a logically equivalent formula  $\psi \in Form^{\{\downarrow\}}$ , then we have proved that  $\{\downarrow\}$  is adequate since all boolean functions can be implemented with  $\{\downarrow\}$  as well.

If we take  $\varphi = (\neg\theta)$  we can write the same function as  $\psi = (\theta \downarrow \theta)$  which is equivalent from the truth tables of  $\downarrow$  and  $\neg$ .

If we take  $\varphi = (\theta_1 \vee \theta_2)$ , since  $\downarrow$  means "not or" it is equivalent to  $\neg(\theta_1 \downarrow \theta_2)$ , so using the previous result we can write the same function as  $\psi = ((\theta_1 \downarrow \theta_2) \downarrow (\theta_1 \downarrow \theta_2))$  and it is equivalent from the truth tables of  $\downarrow$  and  $\vee$ .

Recursively making these replacements down to the proposition level produces a formula in  $Form^{\{\downarrow\}}$  that is equivalent to the given one, so by the previous argument  $\{\downarrow\}$  is an adequate set.

- $\{|\}$  is adequate. We know that the set  $\{\neg, \wedge\}$  is adequate, so all boolean functions can be implemented with  $\{\neg, \wedge\}$ . By the same argument as for  $\{\downarrow\}$ , if we show that we can translate any formula  $\varphi \in Form^{\{\neg, \wedge\}}$  to a logically equivalent formula  $\psi \in Form^{\{|\}$ , then we have proved that  $\{|\}$  is adequate since all boolean functions can be implemented with  $\{|\}$  as well.

If we take  $\varphi = (\neg\theta)$  we can write the same function as  $\psi = (\theta|\theta)$  which is equivalent from the truth tables of  $|\$  and  $\neg$ .

If we take  $\varphi = (\theta_1 \wedge \theta_2)$ , since  $|\$  means "not and" it is equivalent to  $\neg(\theta_1|\theta_2)$ , so using the previous result we can write the same function as  $\psi = ((\theta_1|\theta_2)|(\theta_1|\theta_2))$  and it is equivalent from the truth tables of  $|\$  and  $\wedge$ .

Recursively making these replacements down to the proposition level produces a formula in  $Form^{\{|\}$  that is equivalent to the given one, so by the previous argument  $\{|\}$  is an adequate set.

5.  $\{false, \rightarrow\}$  is adequate.

We know that the set  $\{\neg, \vee\}$  is adequate, so all boolean functions can be implemented with  $\{\neg, \vee\}$ . If we can prove that we can translate any formula  $\varphi \in Form^{\{\neg, \vee\}}$  to a logically equivalent formula  $\psi \in Form^{\{false, \rightarrow\}}$ , then we have proved that  $\{false, \rightarrow\}$  is adequate since all boolean functions can be implemented with  $\{false, \rightarrow\}$  as well.

If we take  $\varphi = (\neg\theta)$  we can write the same function as  $\psi = (\theta \rightarrow false)$  which is equivalent from the truth table of  $\rightarrow$  and  $\neg$ .

If we take  $\varphi = (\theta_1 \vee \theta_2)$ , it is equivalent to  $\neg(\theta_1 \rightarrow \theta_2)$ , so using the previous result we can write the same function as  $\psi = ((\theta_1 \rightarrow false) \rightarrow \theta_2)$  and it is equivalent from the truth tables of  $\rightarrow$  and  $\vee$ .

Recursively making these replacements down to the proposition level produces a formula in  $Form^{\{false, \rightarrow\}}$  that is equivalent to the given one, so by the previous argument  $\{false, \rightarrow\}$  is an adequate set.

## 4 Automata

Recall the definition of a *DFA transition path*  $\rho^*$ :

$$\begin{aligned}\rho^*(s, a) &= \rho(s, a) \text{ for some } s \in S, a \in \Sigma \\ \rho^*(s, ax) &= \rho^*(\rho(s, a), x) \text{ for some } s \in S, a \in \Sigma, x \in \Sigma^*\end{aligned}$$

so by the inductive definition if  $\rho^*(s, x) = t$  then there is a sequence of length(x)+1 states  $s, \dots, t$  in  $A$  such that  $\rho(s_i, a) = s_{i+1}$  for some  $a \in \Sigma$ . Also, recall that  $x \in L(A)$  iff  $\rho^*(s_0, x) \in F$ .

Then,

$$\begin{aligned}L(A) \neq \emptyset &\text{ iff } \exists x \in \Sigma^* \text{ s.t. } x \in L(A) \\ &\text{ iff } \exists x \in \Sigma^* \text{ s.t. } \rho^*(s_0, x) \in F \\ &\text{ iff } \exists \text{ a sequence states (nodes) in } S \text{ } s_0, \dots, s_f \text{ and } s_f \in F \\ &\text{ iff } \exists \text{ a path in } G_A \text{ from } s_0 \text{ to some state in } F\end{aligned}$$

Which completes the proof. The last step in the proof follows from the construction of the graph  $G_A = (S, E)$  since there is an edge in  $E$  for every transition in the DFA, and  $\rho^*(s_0, x) = s_f$  means that for all  $s_i$  in the ordered sequence,  $\rho(s_i, a) = s_{i+1}$  for some  $a \in \Sigma$ , and so by construction the edge  $(s_i, s_{i+1}) \in E$  and by definition the sequence  $s_0, \dots, s_f$  is a path in  $G_A$ .

## 5 Acknowledgment

The homework of Sumit Nain and Hernan Stamati served as the basis for these sample solutions. All errors, omissions or mistakes are entirely the grader's fault. Any student who identifies a mistake and provides a valid correction will receive the usual bounty of  $x \in \mathbb{N}$  bonus percentage points for this homework, where  $x$  is determined by the severity of the mistake.

Typeset with L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>