Tutorial 2

---

1. A *literal* is a propositional variable or its negation. A *clause* is a disjunction of literals such that a literal and its negation are not both in the same clause, for any literal. Similarly, a *cube* is a conjunction of literals such that a literal and its negation are not both in the same cube, for any literal. A propositional formula is said to be in *Conjunctive Normal Form (CNF)* if it is a conjunction of clauses. A formula is said to be in *Disjunctive Normal Form (DNF)* if it is a disjunction of cubes.

   Let P, Q, R, S, T be propositional variables. An example DNF formula is $(P \wedge \neg Q) \vee (\neg P \wedge Q)$, and an example CNF formula is $(P \vee Q) \wedge (\neg P \vee \neg Q)$. Are they semantically equivalent?

   For the propositional formula $(P \vee T) \rightarrow (Q \vee \neg R) \vee \neg(S \vee T)$, find semantically equivalent formulas in CNF and DNF. Show all intermediate steps in arriving at the final result.

   Hint: Use the following semantic equivalences, where we use $\varphi \Leftrightarrow \psi$ to denote semantic equivalence of φ and ψ:

   - $\varphi_1 \rightarrow \varphi_2 \;\Leftrightarrow\; (\neg\varphi_1 \vee \varphi_2)$
   - *Distributive laws:*
     - $\varphi_1 \wedge (\varphi_2 \vee \varphi_3) \;\Leftrightarrow\; (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3)$
     - $\varphi_1 \vee (\varphi_2 \wedge \varphi_3) \;\Leftrightarrow\; (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)$
   - *De Morgan's laws*
     - $\neg(\varphi_1 \wedge \varphi_2) \;\Leftrightarrow\; (\neg\varphi_1 \vee \neg\varphi_2)$
     - $\neg(\varphi_1 \vee \varphi_2) \;\Leftrightarrow\; (\neg\varphi_1 \wedge \neg\varphi_2)$

Solution: Propositional formula $(P \vee T) \rightarrow (Q \vee \neg R) \vee \neg(S \vee T)$.

Transformation steps

1. Eliminate Implications:

$$\neg(P \vee T) \vee \big((Q \vee \neg R) \vee \neg(S \vee T)\big)$$

2. Apply De Morgan's Laws:

$$(\neg P \wedge \neg T) \vee \big((Q \vee \neg R) \vee (\neg S \wedge \neg T)\big)$$

Transformation into DNF
Distribute Conjunction over Disjunction:

$$(\neg P \wedge \neg T) \vee (Q) \vee (\neg R) \vee (\neg S \wedge \neg T)$$

Transformation into CNF
Distribute Disjunction over Conjunction:

$$(\neg P \vee Q \vee \neg R \vee \neg S) \wedge (\neg P \vee Q \vee \neg R \vee \neg T)$$
$$\wedge\, (\neg T \vee Q \vee \neg R \vee \neg S) \wedge (\neg T \vee Q \vee \neg R \vee \neg T)$$

2. Consider the parity function, PARITY : $\{0,1\}^n \mapsto \{0,1\}$, where PARITY evaluates to 1 if and only if an odd number of inputs is 1. In all of the CNFs below, we assume that each clause contains any variable at most once, i.e. no clause contains expressions of the form $p \wedge \neg p$ or $p \vee \neg p$. Furthermore, all clauses are assumed to be distinct.

    1. Prove that any CNF representation of PARITY must have n literals (from distinct variables) in every clause.

       [Hint: Take a clause, and suppose the variable v is missing from it. What happens when you flip v?]

    2. Prove that any CNF representation of PARITY must have $\geq 2^{n-1}$ clauses.

       [Hint: What is the relation between CNF/DNF and truth tables?]

    PARITY is ubiquitous across Computer Science; for example, in Coding Theory (see Hadamard codes), Cryptography (see the Goldreich-Levin theorem), and discrete Fourier Analysis (yes, that is a thing)! PARITY is ubiquitous across Computer Science; for example, in Coding Theory (see Hadamard codes), Cryptography (see the Goldreich-Levin theorem), and discrete Fourier Analysis (yes, that is a thing)!

Solution: Let

$$\text{PARITY} := \bigwedge_{i=1}^{m} \left(\vee_{j=1}^{n_i} \ell_{ij} \right)$$

be the CNF representation of PARITY. We want to prove that $n_i = n$ for every i, and $m \geq 2^{n-1}$.

1. Suppose $n_i < n$ for some i. Negate the entire formula to convert the CNF into a DNF. Now, choose an assignment of literals in the i^{th} cube (of the DNF) such that the cube, and hence the whole DNF formula, evaluates to 1. Now, consider a variable v, which doesn't appear in the i^{th} cube, and flip its value. The LHS then becomes 0. However, the i^{th} clause stays 1, leading to a contradiction.

2. Once again, consider the DNF. Note that a conjunction of n literals is satisfied by a unique assignment of variables, and thus, the clauses in the DNF actually encode the assignments that satisfy the formula. Since PARITY is satisfied by 2^{n-1} clauses, we're done.

3. We have already defined CNF and DNF formulas in Question 1. Recall the definition of *equisatisfiable* formulas taught in class. A **stronger notion of equisatisfiability** is as follows: A formula F with variables $\{f_1, f_2 \cdots f_n\}$ and a formula G with variables $\{f_1, f_2 \cdots f_n, g_1, g_2 \cdots g_m\}$ are **strongly-equisatisfiable** if:

   - For any assignment to the f_is which makes F evaluate to true there exists an assignment to the g_is such that G evaluates to true under this assignment with the same assignment to the f_is.

     $$AND$$

   - For any assignment setting G to true, the assignment when restricted to f_is makes F evaluate to true.

   Why does this help us? A satisfying assignment for G tells us a satisfying assignment for F, and if we somehow discover that G has no satisfying assignments, then we know that F also has no satisfying assignments. In other words, the satisfiability of F can be judged using the satisfiability of G. Satisfiability is a central problem in computer science (for reasons you will see in CS218 later), and such a reduction is very valuable. In particular, for a formula F in k-CNF we look for a formula G that has a small number of literals in each clause, and the total number of clauses itself is not too large - ideally a linear factor of k as compared to F.

   Now consider an r-CNF formula, i.e a CNF formula with r literals per clause. To keep things simple, suppose $r = 2^k$, for some $k > 0$. We will start with the simplest case: a 2^k-CNF formula containing a single clause $C_k = p_0 \vee p_1 \vee p_2 \cdots p_{2^k-1}$ with 2^k literals.

   1. Let's try using 'selector variables' to construct another formula which is strongly-equisatisfiable with C_k. The idea is to use these selector variables to 'select' which literals in our clause are allowed to be false. Use k selector variables $s_1, s_2, s_3 \cdots s_k$ and write a CNF formula that is **strongly-equisatisfiable with** C_k, such that each clause has size $O(k)$.
      [Hint: If you consider the disjunction of $\tilde{s}_1 \vee \tilde{s}_2 \vee \tilde{s}_3 \cdots \tilde{s}_k$, where $\tilde{s}_i$ denotes either s_i or $\neg s_i$, it is true in all but one of the 2^k possible assignments to the s_i variables]
   2. What is the number of clauses in the CNF constructed in the above sub-question? How small can you make the clauses by repeatedly applying this technique?
   3. Consider a k-CNF formula with n clauses. When minimizing the clause size using the above technique repeatedly, what is the (asymptotic) blowup factor in the number of clauses? Observe that the factor is almost linear, upto some logarithmic factors.
      [Hint: Apply the above reduction repeatedly and see how much net blowup occurs]
   4. Can you think of a smarter way to do the reduction that only requires linear blowup?
      [Hint: Consider using different and more auxiliary variables (like the 'selector variables' above) to reduce the size of clauses.]

Solution:

1. The idea is very similar to a MUX (or multiplexer) from digital electronics. We consider the 2^k possible disjunctions: $s_1 \vee s_2 \vee \cdots s_k$; $s_1 \vee s_2 \vee \cdots \neg s_k$ etc. We associate a non-negative integer with each such disjunction, where the k-bit binary encoding of the integer is obtained as $b_1 b_2 \ldots b_k$, where $b_k = 1$ if s_k is in the disjunction and $b_k = 0$ otherwise. For example, the integer associated with the disjunction $(s_1 \vee s_2 \vee \cdots s_k)$ is $11 \cdots 1$ or $2^k - 1$, and the integer associated with $(\neg s_1 \vee \neg s_2 \vee \cdots \neg s_k)$ is $00 \cdots 0$ or 0. Let the disjunction corresponding to the number $0 \leq q \leq 2^k - 1$ in binary be denoted D_q. Consider the CNF formula:

   $$\bigwedge_{i=0} (D_i \vee p_i)$$

If any of the p_is is true, the unique assignment setting D_i to false satisfies all the other $2^k - 1$ clauses. On the other hand if all the p_is are false, any assignment to the s_i variables will dissatisfy atleast one D_q. Thus this formula is strongly equisatisfiable - any assignment setting C_k to true has a satisfying assignment to the s_is, and any assignment setting our formula to true must have one of the p_is true, which makes C_k true.

2. The size of clauses is now $k+1$. The number of clauses is 2^k. Thus, we have obtained an exponential reduction in the size of a clause for a linear factor (the initial size was 2^k blowup in number of clauses. We can reduce the size as long as $\lceil \log_2(\text{size}) \rceil + 1 < \text{size}$, which is true as small as size 3. The size CANNOT be reduced below 3 with this method.

3. We obtain logarithmic reduction for linear blowup. Hence, the factor of blowup will look like $k \cdot \log(k) \cdot \log\log(k) \cdot \log\log\log(k) \cdots \log^{\log^*(k)}(k)$. The factors after the k are all logarithmic factors and can be bounded by a polynomial of any degree > 0.

4. Indeed, we do not need to introduce so many selectors for a clause! We can simply use an indicator s_i to indicate that a clause after p_i is satisfied. Thus, the equisatisfiable formula:

   $$(p_1 \vee p_2 \vee s_2) \wedge (\neg s_2 \vee p_3 \vee s_3) \wedge (\neg s_3 \vee p_4 \vee s_4) \cdots$$

   works and has a linear blowup! Think about Tseitin encoding covered in class.

   However this method also fails to lower the size below 3. Indeed we should suspect that it is not possible to reduce the size below 3, because 2-SAT has a linear time solution while 3-SAT is NP-Complete!

4. **[Take-away question – solve in your rooms]**

In this question we will view the set of assignments satisfying a set of propositional formulae as a language and examine some properties of such languages.

Let $\mathbf{P}$ denote a countably infinite set of propositional variables $p_0, p_1, p_2, \ldots$. Let us call these variables positional variables. Let Σ be a countable set of formulae over these positional variables. Every assignment $\alpha : \mathbf{P} \to \{0,1\}$ to the positional variable can be uniquely associated with an infinite bitstring w, where $w_i = \alpha(p_i)$. The language defined by Σ - denoted by $L(\Sigma)$ - is the set of bitstrings w for which the corresponding assignment α, that has $\alpha(p_i) = w_i$ for each natural i, satisfies Σ, that is, for each formula $F \in \Sigma$, $\alpha \models F$. In this case, we say that $\alpha \models \Sigma$. Let us call the languages definable this way PL-definable languages.

(a) Show that PL-definable languages are closed under countable intersection, ie if $\mathcal{L}$ is a countable set of PL-languages, then $\bigcap_{L \in \mathcal{L}} L$ is also PL-definable.

(b) Show that PL-definable languages are closed under finite union, ie if $\mathcal{L}$ is a finite set of PL-languages, then $\bigcup_{L \in \mathcal{L}} L$ is also PL-definable.

[Hint: Try proving that the union of two PL-definable languages is PL-definable. The general case can then be proven via induction. If $\mathbf{F} = \{F_1, F_2, \ldots\}$ and $\mathbf{G} = \{G_1, G_2, \ldots\}$ are two countable sets of formulae, then an infinite bitstring $w \in L(\mathbf{F}) \cup L(\mathbf{G})$ if and only if either $w \models$ every F_i or $w \models$ every G_i.]

Solution:

a) Let $\mathcal{S}$ denote the family of sets of propositional formulae such that $\mathcal{L} = \{L(\sigma) : \sigma \in \mathcal{S}\}$. Consider $\Sigma = \bigcup_{\sigma \in \mathcal{S}} \sigma$. Since each $\sigma \in \mathcal{S}$ is countable, and a countable union of countable sets is countable, Σ is countable as well. Now, for any infinite bitstring w, $w \in L(\Sigma)$ if and only if $w \models F$ for each $F \in \sigma$, for each $\sigma \in \mathcal{S}$ (we abuse notation and denote the assignment corresponding to w by w). This happens if and only if $w \in L(\sigma)$ for each $\sigma \in \mathcal{S}$, ie $w \in \bigcap_{L \in \mathcal{L}} L$.

b) We show that the union of two PL-definable languages is PL-definable. The general case can then be handled via induction. Let $\mathbf{F}$ and $\mathbf{G}$ be two countable sets of formulae. Let $\mathbf{H} = \{F \vee G : F \in \mathbf{F}, G \in \mathbf{G}\}$ (note that this set is countable since $\mathbf{F}$ and $\mathbf{G}$ are countable). An infinite bitstring w lies in $L(\mathbf{H})$ if and only if, for every $F \in \mathbf{F}$ and $G \in \mathbf{G}$, $w \models F \vee G$. Now, if $w \in L(\mathbf{F})$, then $w \models F$ for every $F \in \mathbf{F}$, hence, $w \models F \vee G$ for every $F \in \mathbf{F}$ and every $G \in \mathbf{G}$ and so $w \in L(\mathbf{H})$. Similarly, if $w \in L(\mathbf{G})$, then $w \in L(\mathbf{H})$, ie for any $w \in L(\mathbf{F}) \cup L(\mathbf{G})$, we have $w \in L(\mathbf{H})$. Let us show that these are the only bitstrings in $L(\mathbf{H})$. Assume some infinite bitstring w is such that $w \notin L(\mathbf{F})$ and $w \notin L(\mathbf{G})$. This means that there is some $F \in \mathbf{F}$ and $G \in \mathbf{G}$ such that $w \not\models F$ and $w \not\models G$. This means that $w \not\models F \vee G$, which is a member of $\mathbf{H}$. Hence $w \notin L(\mathbf{H})$. This means that for any infinite bitstring w, $w \in L(\mathbf{H})$ if and only if $w \in L(\mathbf{F}) \cup L(\mathbf{G})$, ie $L(\mathbf{H}) = L(\mathbf{F}) \cup L(\mathbf{G})$, showing that $L(\mathbf{F}) \cup L(\mathbf{G})$ is PL-definable.