CS 208 : Automata Theory and Logic Spring 2024

Practice Problems 1

0. A Bit of Warm-Up

(a) A student has written the following propositional logic formula over variables x, y, z.

$$x \to \big((y \to \bot) \lor (\top \to z)\big)$$

Is it possible to write a semantically equivalent formula using only the operators $\neg$ and $\lor$?

(b) Consider the following CNF formula over propositional variables p, q, r, s, t.

$$\varphi(p, q, r, s, t) = (p \lor q \lor r) \land (\neg p \lor q) \land (\neg q \lor r) \land (\neg r \lor s) \land (\neg s \lor t) \land (\neg t \lor p)$$

Write a semantically equivalent DNF formula with only one cube. Try to avoid application of the distributive laws, as this can lead to a blow-up of intermediate formula sizes.

(c) We wish to show that $\varphi \models (\neg p \lor (q \land r \land s \land t))$. One way to show this is to show that $\varphi \land \neg(\neg p \lor (q \land r \land s \land t))$ is unsatisfiable. Since $\neg(\neg p \lor (q \land r \land s \land t)) = (p) \land (\neg q \lor \neg r \lor \neg s \lor \neg t)$, our problem reduces to showing unsatisfiability of $\varphi \land (p) \land (\neg q \lor \neg r \lor \neg s \lor \neg t)$. Show using repeated application of resolution rules (and nothing else) that the above formula is indeed unsatisfiable.

Solution:

1. Indeed, it's possible. Here are suggested steps.
   - Write every implication $\alpha \to \beta$ as $(\neg \alpha \lor \beta)$. This gives us the formula
   $$\neg x \lor \big((\neg y \lor \bot) \lor (\neg \top \lor z)\big)$$
   - Simplify every $(\alpha \lor \bot)$ to α. Similarly, $\neg \top$ can be replaced by $\bot$. This gives us
   $$\neg x \lor \big((\neg y) \lor (z)\big)$$
   - Finally, apply DeMorgan's laws to replace every $\alpha \lor \beta$ with $\neg(\neg \alpha \land \neg \beta)$. This gives
   $$\neg\big(x \land (y \land \neg z)\big)$$
2. Notice that φ can be written as
   $$(p \lor q \lor r) \land (p \to q) \land (q \to r) \land (r \to s) \land (s \to t) \land (t \to p)$$
   The five implications above force p, q, r, s, t to have the same value in any satisfying assignment of φ (try to convince yourself why this is the case). Therefore, for φ to be satisfied, we must have at least one of p, q, r to be true (because of the first clause of φ), and all of p, q, r, s, t must have the same value (because of the remaining clauses of φ). This implies that the only satisfying assignment of φ is $p = q = r = s = t = 1$. Hence, the semantically equivalent DNF formula is $p \land q \land r \land s \land t$.

3. We don't even need all clauses of φ to show unsatisfiability by resolution. Here are suggested resolution steps (alternative resolution steps are also possible).

Clause id	Clause	How obtained
1	$\neg p \vee q$	Given
2	p	Given
3	q	Resolvent of 1, 2
4	$\neg q \vee \neg r \vee \neg s \vee \neg t$	Given
5	$\neg r \vee \neg s \vee \neg t$	Resolvent of 3, 4
6	$\neg q \vee r$	Given
7	r	Resolvent of 3, 6
8	$\neg s \vee \neg t$	Resolvent of 7,5
9	$\neg r \vee s$	Given
10	s	Resolvent of 7, 9
11	$\neg t$	Resolvent of 8, 10
12	$\neg s \vee t$	Given
13	t	Resolvent of 10, 12
14	Empty clause ()	Resolvent of 11, 13

1. Core of unsatisfiability

A set $\mathcal{S}$ of propositional logic formulae is said to form a *minimal unsatisfiable core* if $\mathcal{S}$ is unsatisfiable (i.e. there is no assignment of values to variables that satisfies all formulas in $\mathcal{S}$), but every proper subset of $\mathcal{S}$ is satisfiable.

1. Given an unsatisfiable set $\mathcal{S}$ of propositional logic formulae, the minimal unsatisfiable core may not be unique. Give an example where $\mathcal{S}$ has at least two minimal unsatisfiable cores $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$.
2. Show that for every $n > 0$, we can find a set of n propositional logic formulae that form a minimal unsatisfiable core. Thus, we can have minimal unsatisfiable cores of arbitrary finite size.

Solution:

1. This is fairly straightforward. Consider the set $\mathcal{S} = \{p, p \rightarrow q, \neg q, p \rightarrow r, \neg r\}$. The two minimal unsatisfiable cores are $\mathcal{C}_1 = \{p_1, p_1 \rightarrow p_2, \neg p_2\}$ and $\mathcal{C}_2 = \{p_1, p_1 \rightarrow p_3, \neg p_3\}$

2. The set of formulas $\mathcal{S}_n = \{p_1, p_1 \rightarrow p_2, p_2 \rightarrow p_3, \ldots p_{n-2} \rightarrow p_{n-1}, \neg p_{n-1}\}$ satisfies the condition of the question. Why is the entire set $\mathcal{S}_n$ is unsatisfiable? This should be easy to figure out.

   Every proper subset of $\mathcal{S}_n$ either has p_1 missing, $\neg p_3$ missing or some implication $p_i \rightarrow p_{i+1}$ missing for $1 \leq i \leq n-2$. We show that in each of these cases, the remaining subset of formulas is satisfiable.

   - If p_1 is missing, all the other formulas are satisfied by setting $p_2 = \ldots = p_{n-1} = 0$.
   - If $\neg p_{n-1}$ is missing, all the other formulas are satisfied by setting $p_1 = \ldots = p_{n-2} = 1$.
   - If $p_i \rightarrow p_{i+1}$ is missing for $1 \leq i \leq n-2$, then all other formulas are satisfied by setting $p_1 = \ldots p_i = 1$ and $p_{i+1} = \ldots p_{n-1} = 0$.

2. Logical interpolation

Let φ and ψ be propositional logic formulas such that $\models \varphi \rightarrow \psi$ (i.e. $\varphi \rightarrow \psi$ is a tautology). Let $Var(\varphi)$ and $Var(\psi)$ denote the set of propositional variables in φ and ψ respectively. Show that there exists a propositional logic formula ζ with $Var(\zeta) \subseteq Var(\varphi) \cap Var(\psi)$ such that $\models \varphi \rightarrow \zeta$ and $\models \zeta \rightarrow \psi$. This result is also known as *Craig's interpolation theorem* as applied to propositional logic. The formula ζ is called an *interpolant* of φ and ψ.

As a specific illustration of the above result, consider the formulas $\varphi = ((p \rightarrow q) \wedge (q \rightarrow \neg p))$ and $\psi = (r \rightarrow (p \rightarrow s))$, where p, q, r, s are propositional variables. Convince yourself that $\models \varphi \rightarrow \psi$ holds in this example. Note that $Var(\varphi) = \{p, q\}$ and $Var(\psi) = \{p, r, s\}$. Let $\zeta = \neg p$. Then $Var(\zeta) \subseteq Var(\varphi) \cap Var(\psi)$. Convince yourself that $\models \varphi \rightarrow \zeta$ and $\models \zeta \rightarrow \psi$ hold in this example.

Solution: Given a formula φ and a variable $v \in Var(\varphi)$, let $\varphi[v = \top]$ denote the formula obtained by replacing all occurrences of v in φ with $\top$, and then simplifying the resulting formula. By simplification, we mean the obvious ones like $\alpha \wedge \top = \alpha$, $\alpha \vee \top = \top$, $\alpha \wedge \neg \top = \bot$, $\alpha \vee \neg \top = \alpha$ for all sub-formulas α of φ. In a similar manner, we define $\varphi[v = \bot]$.

Note that $Var(\varphi[v = \top]) = Var(\varphi) \setminus \{v\}$ and similarly for $\varphi[v = \bot]$.

Now define the formula $\zeta = \bigvee_{v \in Vars(\varphi) \setminus Vars(\psi)} \bigvee_{a \in \{\bot, \top\}} \varphi[v = a]$. Notice that $Vars(\zeta) \subseteq Vars(\varphi) \cap Vars(\psi)$.

You should now be able to argue that (i) $\varphi \models \zeta$, and (ii) $\zeta \models \psi$. Proving (i) should be straightforward from the definition of ζ. To prove (ii), take any satisfying assignment of ζ. By definition of ζ, this gives an assignment of truth values to variables in $Vars(\varphi) \cap Vars(\psi)$ such that this assignment can be augmented with an assignment of truth values to variables in $Vars(\varphi) \setminus Vars(\psi)$ to satisfy the formula φ. However, since $\varphi \rightarrow \psi$ is a tautology, this (augmented) assignment also satisfies ψ. Recalling that satisfaction of ψ cannot depend on the assignment of truth values to variables not present in ψ, we conclude that the assignment of truth values to variables in $Vars(\varphi) \cap Vars(\psi)$ itself satisfies the formula ψ. Hence $\zeta \models \psi$.

3. DPLL with horns

The Horn-Sat problem entails checking the satisfiability of Horn formulas. We say a formula is a Horn formula if it is a conjunction ($\wedge$) of Horn clauses. A Horn clause has the form $\phi_1 \to \phi_2$ where ϕ_1 is either $\top$ or a conjunction ($\wedge$) of one or more propositional variables. ϕ_2 is either $\bot$ or a **single** propositional variable. In this context, answer the following questions

1. Let's try to solve the Horn-Sat problem using DPLL. We can convert each Horn clause into a CNF clause by simply rewriting $(a \to b)$ as $(\neg a \vee b)$, which preserves the semantics of the formula. The resultant formula is in CNF.
   We have seen in class that if DPLL always chooses to assign 0 to a decision variable before assigning 1 (if needed) to the variable, then DPLL will never need to backtrack when given a Horn formula encoded in CNF as input.
   Suppose our version of DPLL does just the opposite, i.e. it always assigns 1 to a decision variable before assigning 0 (if needed). How many backtracks are needed if we run this version of DPLL on the (CNF-ised version of) following Horn formulas, assuming DPLL always chooses the unassigned variable with the smallest subscript when choosing a decision variable?

   (a)
   $$\Big((\bigwedge_{i=0}^{n-1} x_i) \to x_n\Big) \wedge \bigwedge_{i=0}^{n-1}(x_n \to x_i) \wedge \Big((\bigwedge_{i=0}^{n} x_i) \to \bot\Big)$$

   (b)
   $$\bigwedge_{i=0}^{n-1} \Big((x_i \to x_{n+i}) \wedge (x_{n+i} \to x_i) \wedge (x_i \wedge x_{n+i} \to \bot)\Big)$$

2. Since we have a polynomial time formula for Horn-Sat, the next question is if it will allow us to solve the Boolean-SAT problem in polynomial time. Sadly, this is not the case. Find a boolean function that cannot be expressed by a Horn formula. Prove that no Horn formula can represent the given boolean function.

Solution:

1. It's best to write out the implications as clauses when you are trying to apply DPLL.

   In problem (a), no unit clauses or pure literals are obtained until all of x_0 through x_{n-1} are assigned the value 1, one at a time. Once x_{n-1} is assigned 1, unit propagation leads to a conflict – x_n must be assigned both 1 and 0 by unit propagation. This causes a backtrack, which ends up setting x_{n-1} to 0. Once this happens, unit propagation assigns the value 0 to x_n, resulting in the partial assignment $x_{n-1} = x_n = 0$, which satisfies all clauses. Therefore, there is exactly one backtrack.

   In part (b), every time a variable x_i for $0 \le i \le n-1$ is assigned 1, unit propagation causes x_{n+i} to be in conflict (must be assigned both 0 and 1). This induces a backtrack that sets x_i to 0, followed by unit propagation setting x_{n+i} to 0. DPLL will then choose x_{i+1} as the next decision variables, assign it 1 and the above process repeats. So, in this case, DPLL will incur n backtracks, one for each of $x_0, \ldots x_{n-1}$.

2. $F = (a \vee b)$. Any formula ϕ which is equivalent to a Horn formula has the following property: if v_1 and v_2 are two valuations that make ϕ evaluate to $\top$, then the valuation $v_1 \wedge v_2$ also makes ϕ evaluate to $\top$. Now consider $F = (a \vee b)$. F is satisfied by $(\top, \bot)$ and $(\bot, \top)$ but not $(\bot, \bot)$. Hence, no Horn Formula can represent F.

4. A Game of Sudoku

Sudoku is a logic-based, combinatorial number-placement puzzle. In classic Sudoku, the objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contains all of the digits from 1 to 9 (and as one can naturally deduce; contain each of the digits exactly once). The puzzle setter provides a partially completed grid, which has a single solution for a well-posed puzzle. Encode the puzzle as a CNF formula. You are free to play around with different encodings; use auxiliary variables and attempt to make succinct (read; "optimised") formulae.

Solution: Let $p(i, j, k)$ be a propositional variable that asserts if the cell in row i and column j has the value n. One can clearly note that $1 \leq i, j, k \leq 9$. We encode the following constraints

- Every cell contains at least one number:

$$\phi_1 = \bigwedge_{i=1}^{9} \bigwedge_{j=1}^{9} \bigvee_{k=1}^{9} p(i, j, k)$$

- Every cell contains at most one number:

$$\phi_2 = \bigwedge_{i=1}^{9} \bigwedge_{j=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{y=x+1}^{9} (\neg p(i, j, x) \vee \neg p(i, j, y))$$

- Every row contains every number:

$$\phi_3 = \bigwedge_{i=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{j=1}^{9} p(i, j, k)$$

- Every column contains every number:

$$\phi_4 = \bigwedge_{j=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{i=1}^{9} p(i, j, k)$$

- Every 3×3 box contains every number:

$$\phi_5 = \bigwedge_{r=0}^{2} \bigwedge_{s=0}^{2} \bigwedge_{n=1}^{9} \bigvee_{i=1}^{3} \bigvee_{j=1}^{3} p(3r + i, 3s + j, n)$$

The final formula is as follows

$$\phi = (\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5)$$

5. Poring over proofs

A student has given the following proof of $\top \vdash x \rightarrow \neg x$ What are the sources of problem in this proof (else we would be in serious trouble with true being equivalent to false).

```
1. top
    --------------------------------------------------------
2.   | x                 assumption                         |
     |  --------------------------------------------------  |
3.   |  | neg x          assumption                      |  |
4.   |  | bot            bot introduction rule on 2 and 3 |  |
     |  --------------------------------------------------  |
5.   | neg x             bot elimination rule on 4          |
6.   | bot               bot intro rule on 2 and 5          |
    --------------------------------------------------------
7. neg x                 neg intro rule on 2 -- 6
    ----------------------------------------------
8.   | x                 assumption               |
9.   | bot               bot intro rule on 7 and 8 |
10.  | neg x             bot elim rule on 9       |
    ----------------------------------------------
11.x -> neg x            impl intro rule on 8 to 10
```

Solution: Thankfully, there is an error in the proof.

Step 5 infers $\neg x$ outside the inner box by applying $\perp$-elimination rule on the result of Step 4. However, $\perp$ was derived in the scope of the inner box in Step 4. Deriving $\neg x$ inside the inner box after Step 4 would have been fine, but $\perp$-elimination doesn't allow us to infer $\neg x$ outside the scope of the inner box.

6. The Resolution Proof System

Consider the formula $\bigoplus_{i=1}^{n} x_i$, where $\oplus$ represents xor. It can be shown by induction that this is semantically equivalent to the PARITY function that evaluates to true if and only if an odd number of variables are assigned true.

Show that $\bigwedge_{\substack{S\subseteq\{1,2\ldots n\}\\ |S|=1 \mod 2}} \left(\bigvee_{i\in S} x_i \vee \bigvee_{j\notin S} \neg x_j \right)$ is the only CNF equivalent to φ, upto adding tautological clauses and repeating variables.

Hint: If two CNFs φ and ψ are equivalent, then for every clause α in φ, we have $\psi \vdash \alpha$ and for every clause β in ψ, we have $\varphi \vdash \beta$, where the proofs used are resolution proofs.

As a refresher, the resolution proof system is a system of proof rules for CNFs that is both sound and complete, ie if φ and ψ are CNFs, then $\varphi \models \psi$ if and only if $\varphi \vdash \psi$. We say $\varphi \vdash \psi$ iff for every clause c in ψ, we have $\varphi \vdash c$. The proof rules that can be applied in resolution proofs are:

1. (*Assumption*) For every clause c in φ, $\varphi \vdash c$
2. (*Resolution*) If we have $\varphi \vdash p \vee c_1$ and $\varphi \vdash \neg p \vee c_2$ for any clauses c_1 and c_2 and propositional variable p, then we can deduce $\varphi \vdash c_1 \vee c_2$.
3. (*Or Introduction*) For any clauses c_1 and c_2, if we have $\varphi \vdash c_1$, then we can deduce $\varphi \vdash c_1 \vee c_2$

Other than these, we are implicitly allowed to reorder any of the literals within a clause and any of the clauses within a CNF, and we can remove tautological clauses from the CNF (these are the clauses that contain a variable as well as it's negation).

Solution: An assignment α satisfies the formula if and only if the number of variables set to 1 by α is odd. An assignment does not satisfy the CNF given in the problem (call it φ) if and only if the some clause is falsified. This occurs if and only if an even number of variables are set to true. Therefore, an assignment satisfies φ if and only if an odd number of variables are set to 1. Therefore, φ in the question is semantically equivalent to $\bigoplus_{i=1}^{n} x_i$.

Assume there is some other CNF ψ also equivalent to $\bigoplus_{i=1}^{n} x_i$, where there are no tautological clauses in ψ and no repeated variables in a single clause. φ and ψ will be equivalent CNFs, ie $\varphi \models \psi$ and $\psi \models \varphi$.

From the first condition, we can conclude that for any clause c in ψ, we have $\varphi \vdash c$ by a resolution proof.

The **key idea** here is to note that when two clauses of χ are resolved, only tautological clauses result. Say there are clauses $c_1 = x \cup C_1$ and $c_2 = \neg x \cup C_2$ being resolved with respect to the variable x. There must be some other variable y such that y is present in C_1 and $\neg y$ is present in C_2 (or vice versa). This is as the number of non-negated variables in each clause in χ must be odd. Therefore, the resolvent $C_1 \cup C_2$, will contain both y and $\neg y$ and will therefore be a tautology.

Also note that since each clause of φ already contains every variable, applying *or introduction* on a clause either leaves it unchanged, or results in a tautology.

This means that the only proof rule that can be used to get non-tautological clauses is *Assumption*. Hence, if $\varphi \vdash c$, c must have been a clause of φ in the first place. This means that all the clauses in ψ must have already been present in φ.

Now, we must also have $\psi \vdash c$ for every clause c in φ. Since every clause in ψ is also a clause in φ, we can again conclude that *resolution* on them also produces only tautologies, and similarly *or introduction* is useless as well. Therefore, if $\psi \vdash c$, then c must have been a clause of ψ. Therefore, every clause in φ must also be present in ψ.

Hence the clauses of ψ and φ must be the same, ie ψ and φ must be the same, meaning that the CNF obtained is unique.

7. A flavour of DFAs

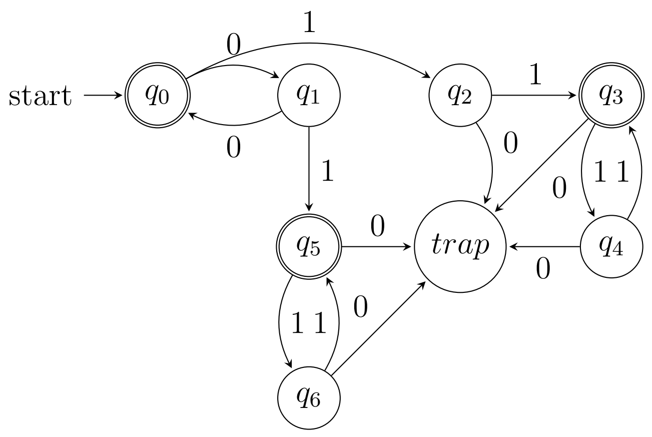
Consider a DFA M with the set of states Q, alphabet $\Sigma = \{0, 1\}$, transition function δ, and accepting state F.

Draw the state diagram for the DFA M that accepts the language $L = \{0^n1^m \mid m + n \text{ is even}\}$. The DFA should recognize strings where the total number of '0's and '1's is even. Provide the diagram along with the state transitions.

Note: $L(M)$ represents the language accepted by DFA M, and $\sim_L$ is the equivalence relation induced by the language.

Solution:

The state diagram for the DFA M is as follows:



where the set of states $Q = \{q_0, q_1, \ldots q_6, trap\}$, alphabet $\Sigma = \{0, 1\}$, transition function δ, and accepting state $F = \{q_0, q_5, q_3\}$ and initial state is q_0.

State	Input 0	Input 1
q_0	q_1	q_2
q_1	q_0	q_5
q_2	trap	q_3
q_3	trap	q_4
q_4	trap	q_3
q_5	trap	q_6
q_6	trap	q_5

Explanation for the states:

- q0: Even no of 0's
- q1: Odd no of 0's
- q2: Odd no of 1's after even 0's
- q3: Even no of 1's
- q4: Odd no of 1's
- q5: Odd no of 1's after odd 0's
- q6: Even no of 1's
- trap: If 0's seen after 1's

8. More solved problems on DFA

Please look at some solved examples from the textbook *Introduction to Automata Theory, Languages and Computation* by J.E. Hopcroft, R. Motwani and J.D. Ullman (2nd or later editions). For example, you can look at Examples 2.2. 2.4, 2.5 to get a feel of DFAs for simple languages.