Harvard University
Computer Science 20
Problem Set 1

PROBLEM 1

A half dozen different operators may appear in propositional formulas, but just $\wedge$, $\vee$, and $\neg$ are enough to express every proposition. That is because each of the operators is equivalent to a simple formula using only these three operators. For example, $A \to B$ is equivalent to $\neg A \vee B$. So all occurrences of $\to$ in a formula can be replaced using just $\neg$ and $\vee$.

We already know that $A \iff B$ is equivalent to $(A \wedge B) \vee (\neg A \wedge \neg B)$ and $A \oplus B$ is equivalent to $(A \wedge \neg B) \vee (\neg A \wedge \neg B)$ by definition.

(A) Prove that we don't even need $\wedge$, that is, write a proposition using **only** $\vee$ and $\neg$ that is equivalent to $A \wedge B$. Prove your answer.

In parts B and C you will prove that we can get by with the single operator NAND. NAND is written $\uparrow$, and is defined as $\neg(A \wedge B)$. To prove we only need $\uparrow$, all you need to do is construct propositions using *only* $\uparrow$ that are logically equivalent to $\neg A$ and $A \vee B$. Nota bene: Because NAND is both sufficient and easy to build in a digital circuit, in practice it is often actually the case that NAND is the only operator.

(B) For this part, write a proposition using only $\uparrow$ and A that is equivalent to $\neg A$. Prove the equivalence of your proposition.
(C) For this part, write a proposition using only $\uparrow$ that is equivalent to $A \vee B$. Prove the equivalence.

Solution.
(A)
$A \wedge B$ can be expressed without using the $\wedge$ logical connective as $\neg(\neg A \vee \neg B)$. This is proven by the truth table constructed below, which demonstrates identical truth values in the columns representing $A \wedge B$ and $\neg(\neg A \vee \neg B)$.

A	B	$\neg A$	$\neg B$	$\neg(\neg A \vee \neg A)$	$A \wedge B$
T	T	F	F	T	T
T	F	F	T	F	F
F	T	T	F	F	F
F	F	F	T	F	F

(B)
$A \uparrow B$ is equivalent to $\neg A$. The equivalence chain below proves their equivalence.
$A \uparrow B \equiv \neg(A \wedge B)$
$A \uparrow A \equiv \neg(A \wedge A)$
$A \uparrow A \equiv \neg(A)$
$A \uparrow A \equiv \neg A$

(C) Equivalence chain proves equivalence between $A \uparrow B$ and $\neg(\neg A \wedge \neg B)$
$A \uparrow B \equiv \neg(A \wedge B)$
$A \uparrow B \equiv \neg A \vee \neg B$: DeMorgan's law
$(A \uparrow A) \uparrow B \equiv \neg\neg A \vee \neg B$: equivalence $(A \uparrow A \equiv \neg A)$ proven in part 1B
$(A \uparrow A) \uparrow (B \uparrow B) \equiv \neg\neg A \vee \neg\neg B$: equivalence $(A \uparrow A \equiv \neg A)$ proven in part 1B
$(A \uparrow A) \uparrow (B \uparrow B) \equiv A \vee B$: double negation

PROBLEM 2

Let x and y be fixed (but unknown) integers. Let $z = x * y$. Let $P =$"x is even", $Q =$ "y is even", and $R =$ "z is even".

Translate the following sentences to propositional logic:

(A) If the product of x and y is even, then at least one of x and y is even
(B) z is even, but x is odd
(C) z is even if x is even
(D) z is even only if x and y are even
(E) z is odd unless x or y is even
(F) For each of the above propositions, determine whether it is true or false or whether it depends on the values of x, y, and z.

Solution.
(A) $R \to P \vee Q$: True
(B) $R \wedge \neg P$: depends on variables
(C) $P \to R$: True
(D) $R \leftrightarrow P \wedge Q$: False
(E) $\neg(P \vee Q) \to \neg R$: True

PROBLEM 3

For each of the following propositions, if it is a tautology or it is unsatisfiable, prove it with a truth table. If it is satisfiable, identify a satisfying assignment.
(A) $(P \vee Q) \vee (Q \rightarrow P)$
(B) $(P \rightarrow Q) \rightarrow P$
(C) $P \rightarrow (Q \rightarrow P)$

Solution.
(A) Satisfiable by (P:T, Q:T) & (P:T, Q:F)

P	Q	$(P \vee Q)$	$(Q \rightarrow P)$	$(P \vee Q) \wedge (Q \rightarrow P)$
T	T	T	T	T
T	F	T	T	T
F	T	T	F	F
F	F	F	T	F

(B) Satisfiable by (P:T, Q:T) & (P:T, Q:F)

P	Q	$(P \rightarrow Q)$	$(P \rightarrow Q) \rightarrow P$
T	T	T	T
T	F	F	T
F	T	T	F
F	F	T	F

(C) $(P \rightarrow (Q \rightarrow P)$ is a Tautology

P	Q	$(Q \rightarrow P)$	$(P \rightarrow (Q \rightarrow P)$
T	T	T	T
T	F	T	T
F	T	F	T
F	F	T	T

PROBLEM 4

Consider a proposition of n variables. Here is an algorithm for checking for satisfiability: Generate a truth table for the proposition. Check if all lines of the truth table are false. If not, the proposition is satisfiable. Why is this algorithm exponentially costly? **Solution.**

As each new variable is added the cases which need to be covered grows 2 fold, such that the number of cases/rows that need to be added to the table can be modelled by 2^n. This makes the algorithm's cost exponential because it is executed on an exponentially growing input. $\binom{2}{1}^n = 2^n$

PROBLEM 5

THIS PROBLEM SHOULD BE SUBMITTED IN CARNAP. ONLY ANSWERS SUBMITTED IN CARNAP WILL BE ACCEPTED.

Convert the following proposition to DNF with a justification for each step: $(P \to Q) \land (\neg(Q \lor \neg R) \lor (P \land \neg S))$

Solution.

solution-6.png

PROBLEM 6

Consider the formula:

$(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \ldots \vee (p_n \wedge q_n)$

where $n \geq 2$ and the p_i and q_i are propositional variables. This proposition has n clauses and length $4n - 1$ if we count each propositional variable and operator as adding 1 to the length and we ignore the parentheses.

(A) First, write the CNF version of the proposition when $n = 2$. Give a justification for each step. **THIS SUBPROBLEM SHOULD BE SUBMITTED IN CARNAP. ONLY ANSWERS SUBMITTED IN CARNAP WILL BE GRADED.**
(B) When you distribute the and over the or, what happens to the number of clauses?
(C) How long is your conjunctive normal form of this formula, using the same conventions as in the problem statement?
(D) For general n, how many clauses are in the conjunctive normal form as a function of n? How long is it?

Solution.
(A)

problem7.png

(B) The number of clauses doubles
(C) The CNF contains 4 clauses. (Sorry, Im not sure what is meant by "same conventions as in the problem statement")
(D) n^2 where n is the number of clauses in the DNF version.

PROBLEM 7

Consider the following algorithm for checking satisfiability: Put the formula into disjunctive normal form and then check to see if all of the disjuncts are contradictions (containing both a variable and its complement). If not, then it is satisfiable. Why is this algorithm exponentially costly? **Solution.**

In the process of building the disjunctive formula, we are forced to consider both the variable and its negation when creating conjunctions. As a result we are forced to consider two possibilities per variable, with n variables, which causes the complexity of our equation to grow at a rate of approximately 2^n where n represent the number of variables. An example of this would be in the distributing of $\wedge$ over $\vee$ operators (when converting from a "CNF-esque" form of the formula), causing the number of terms to grow exponentially.