```
-----------------------------------------------------------------------------------------------
Logic in Computer Science, October 31st, 2023. Time: 1h45min. No books or lecture notes allowed.
-----------------------------------------------------------------------------------------------
 - Insert your answers on the dotted lines ... below, and only there.
 - When finished, upload this file with the same name: exam.txt
 - Use the text symbols:     &     v    -       ->          |=        A       E
     for                    AND   OR   NOT   IMPLIES   "SATISFIES"  FORALL  EXISTS  etc.,  like in:
    I |=  p & (q v -r)     (the interpretation I satisfies the formula p & (q v -r)  ).
   You can write  not (I |= F)  to express "I does not satisfy F",  or
                  not (F |= G)  to express "G is not a logical consequence of F"
   Also you can use subindices with "_". For example write x_i to denote x-sub-i.
-----------------------------------------------------------------------------------------------


Problem 1. (2.5 points).

1a) Is it true that if F -> G is a tautology then G |= F? Prove using only the definition
    of propositional logic.

>>> Answer:
    No. Counterexample: suppose F is p & -p and G is any satisfiable formula (for example
    p). It is easy to verify that for these F and G the formula F -> G, that is,
    the formula -F v G, is tautology: for any interpretation I, since F is unsatisfiable,
    we have eval_I(-F v G) = max(1-eval_I(F), eval_I(G)) = 1. However, F is not a logical
    consequence of G: there exists at least one model I of G (because G is satisfiable),
    and this I is not a model of F (because F is unsatisfiable).

1b) Is it true that for any two propositional formulas F and G, we have that -F v G is a
    tautology if and only if F |= G? Prove it using only the definition of propositional logic.

>>> Answer:
    This is true.
    -F v G is a tautology                iff  [ by definition of tautology           ]
    AI, I |= -F v G                      iff  [ by definition of |=                  ]
    AI, eval_I(-F v G) = 1               iff  [ by definition of eval_I( v )         ]
    AI, max(eval_I(-F), eval_I(G)) = 1   iff  [ by definition of eval_I( - )         ]
    AI, max(1 - eval_I(F), eval_I(G)) = 1  iff  [ by definition of max and eval      ]
    AI, eval_I(F) = 0 or eval_I(G) = 1   iff  [ by definition of satisfaction       ]
    AI, not I |= F or I |= G             iff  [ by definition of "implies"          ]
    AI, I |= F implies I |= G            iff  [ by definition of logical consequence ]
    F |= G


-----------------------------------------------------------------------------------------------

Problem 2. (2.5 points).

In what follows, you can consider proved, and you can use it in a prove,
the following statement:

  (ST) Let F1,F2,F1',F2' be formulas. If F1 |= F1' and F2 |= F2', then
       F1 & F2 |= F1' & F2' and F1 v F2 |= F1' v F2'.

2a) Let F be a formula where v (or) and & (and) are the only
    connectives.  Show that if in F we replace an occurrence of a
    subformula G with another G' such that G |= G', we obtain a new
    subformula F' such that F |= F' (F' is a logical consequence of
    F) Prove the statement by *induction* on n(F), defined as the number
    of connectives of F minus the number of connectives of G (the
    subformula).

>>> Answer:
    * Base case: If n(F) = 0,
      then necessarily F is identically G. Therefore, if we replace G
      with G', then F' is identically G'. Since G |= G' by hypothesis,
      we have F |= F'.

    * Inductive case: suppose that n(F) > 0, and that the statement is
      true for any formula H such that H contains an occurrence of G
      as a subformula, and n(H) < n(F).
      Since n(F) > 0, G is strictly a subformula of F. Then F must be
      either F1 & F2 or F1 v F2, for certain formulas F1,F2.  Suppose
      F of the form F1 @ F2, for certain formulas F1,F2 and @ in
      {&,v}. Without loss of generality we can also assume that the
      formula G that is replaced by G' occurs as a subformula of F1
      (the case in which G that is replaced by G' occurs as a
      subformula of F2 is analogous). We define F1' as the formula
      resulting from substituting G for G' in F1. As n(F1) < n(F), by
      induction hypothesis F1 |= F1'. By the previous result (ST),
      F1 @ F2 |= F1' @ F2, and since F is F1 @ F2 and F' is F1' @ F2,
      we have F |= F'.

2b) Can the previous result be extended to formulas F where the
    connective - (not) can also appear? Justify the answer.

>>> Answer:
    No. Counterexample: let F be the formula -p, G be the formula p,
    and G' be the formula (p v -p).  Clearly G is a subformula of F,
```

G |= G', because G' is a tautology. So F' is -(p v -p), which is
unsatisfiable. As for interpretation I with I(p) = 0 we have
I |= F and not I |= F', it is not true that F |= F'.

Problem 3. (2.5 points).

3a) What is Horn-SAT? What is its computational complexity? Explain very briefly why.

>>> Answer:
Horn-SAT is the problem of deciding the satisfiability of a set of clauses S
such that all clauses in S are Horn: they have at most one positive literal.
It is polynomial, more precisely linear, because we can decide it simply by
unit propagation of positive unit literals.
For example, given the set of four Horn clauses S = { -r, p, -p v q, -p v -q v r },
by propagation of the positive unit clause p, we get q, and then r, and then
a conflict (the empty clause). Then S is unsat: since unit propagation is correct,
all new units are logical consequences of S.
Another example: S = { -r v -r', p, -p v q, -p v -q v r }. We also propagate
positive units p, q, r, but no empty clause appears. In that case S is sat,
since we get a model I by setting all propagated units to 1 and the rest to 0:
here I(p) = I(q) = I(r) = 1 and I(r') = 0. Indeed then always I is a model of S,
i.e., I |= C, for every non-empty clause C of S that is not a positive unit clause:
if C has propagated, then I |= C; otherwise, since C is Horn, it must have at least
one negative literal -p where p has not been propagated, so I |= C too.

3b) Let S be a set of propositional clauses over a set of n predicate symbols,
and let Res(S) be its closure under resolution. For each one of the following
cases indicate whether Res(S) is infinite or finite, and, if finite, of which size.
Consider that each clause is a set (i.e., no repetitions) of literals.

* If clauses in S have at most two literals.

>>> Answer:
Note that if clauses in S have at most two literals, clauses in Res(S) too.
If there are n predicate symbols, there are 2n literals. There are binomial(2n, 2)
clauses (i.e., subsets) of two literals, plus 2n clauses of 1 literal,
plus the empty clause, which altogether is O(n^2). Therefore, this is the
maximal size of Res(S) in this case.

* Every clause in S has either two literals or is a Horn clause.

>>> Answer:
Starting from Horn clauses and two-literal clauses in S does not help
in bounding the size of Res(S). In fact, from certain S of this form,
by resolution one can obtain any clause: for any clause with a negative
literal -p, such as -p v C, by one resolution step with a two-literal
clause p v q we get the clause q v C, and another step with -q v p we get
the clause p v C, i.e., we can change the sign of any literal of any clause.
Still, for a given set of n predicate symbols, Res(S) is always finite,
because each clause is a subset of the set of 2n literals, so there are
at most 2^(2n) clauses in Res(S).

Problem 4. (2.5 points).

4) Consider the following C++ code snippet:

```
1: int f(int i, int s, const vector<int>& v) {
2:    if (s == 0) return 1;
3:    while (i < 0 and s != 0) ++i;
4:    if (i >= v.size()) return -1;
5:    return v[i];
6: }
```

Let us introduce propositional symbols p, q, r with the following meaning:

```
p:  "i >= 0       at line 5"
q:  "i < v.size() at line 5"
r:  "s == 0"
```

We note that the value of program variable s never changes in the code,
so in the definition of propositional symbol r we do not need to indicate
which line we are referring to.

4a) Give a propositional formula in CNF such that if it is unsatisfiable then we can ensure that
the vector access v[i] at line 5 is correct.

>>> Answer:
At line 5 we have -r, because if s == 0 we would have returned at line 2.
We also have p v r, because after exiting the loop at line 3 we have -(i < 0 & s != 0).
Finally, at line 5 we have q too, because of the statement at line 4.

On the other hand, the vector acces v[i] at line 5 is correct if and only if
0 <= i < v.size(), that is, p & q.

So if we prove that
-r & (p v r) & q |= p & q
then the vector access will be correct.

But this is equivalent to proving that
-r & (p v r) & q & (-p v -q)
is unsatisfiable.

4b) Prove that indeed the access v[i] at line 5 is correct.

>>> Answer:
Let us apply a DPLL-based SAT solver to show that formula

-r & (p v r) & q & (-p v -q)

is unsatisfiable.

The clauses are:

1. -r
2. p v r
3. q
4. -p v -q

Then by applying DPLL we see the formula is indeed unsatisfiable:

```
    up 1         up 3            up 2               fail 4
[] ------> [-r] ------> [-r, q] ------> [-r, q, p] --------> backtrack (but no backtracking is possible, so
unsat)
```

We can also prove unsatisfiability through resolution:

```
          p v r      -p v -q
          ------------------
                r v -q         -r
                -----------------
                        -q           q
                        --------------
                              []
```