After doing this tutorial you should be able to:

1. use the recursive definition to tell if something is a Boolean formula or not,
2. evaluate a given Boolean formula for a given assignment,
3. tell if a given Boolean formula is satisfiable/valid,
4. model facts and reasoning in Boolean logic.

Problem 1. Are the following expressions propositional logic formulas (according to the definition of the syntax, the conventions, as well as the extended definition of the syntax, given in the lecture)?

1. $(p \wedge q)$
2. (p)
3. $p \wedge q$
4. $\neg(p \wedge q)$
5. $(p \oplus q)$
6. $(p \rightarrow q)$
7. $(p \rightarrow (q \vee r))$
8. $\neg\neg\neg\neg\neg p$

Solution 1.

1. Yes.
2. No, we only add parentheses when joining two formulas with a binary connective (e.g. $\wedge$, $\vee$, etc.).
3. Yes, although it uses the convention that we drop outermost parentheses.
4. Yes.
5. No, because the symbol $\oplus$ is not one of the defined connectives, nor in the extended definition that we introduced. However, you may recognise that it usually means "exclusive or". *You* can introduce it as an abbreviation (in the same way as we did in lectures) by saying that we introduce $(F_1 \oplus F_2)$ as an abbreviation of $((F_1 \vee F_2) \wedge \neg(F_1 \wedge F_2))$.
6. Yes. $\rightarrow$ is not part of the original definition, but is part of the extended definition.
7. Yes. $(q \vee r)$ is a formula, and p is a formula, so $(p \rightarrow (q \vee r))$ is a formula (in the extended definition).
8. Yes, just apply the rule that if F is a formula then so is $\neg F$.

Problem 2. A formula G is **valid** if every assignment satisfies G. Valid formulas capture "logical truths". Give some examples of valid formulas.

Solution 2. Some valid formulas are:

- $p \vee \neg p$,
- $(p \wedge q) \leftrightarrow (q \wedge p)$
- $(p \wedge q) \rightarrow (p \vee q)$
- $(p \rightarrow q) \leftrightarrow (\neg p \vee q)$.

Problem 3. This is Exercise 7.10 in Russel and Norvig's *Artificial Intelligence* (2010).

Decide whether each of the following sentences is valid, unsatisfiable, or neither. Verify your answer using truth-tables (or any method you like).

1. $Smoke \rightarrow Smoke$
2. $Smoke \rightarrow Fire$
3. $(Smoke \rightarrow Fire) \rightarrow (\neg Smoke \rightarrow \neg Fire)$
4. $Smoke \vee Fire \vee \neg Fire$
5. $((Smoke \wedge Heat) \rightarrow Fire) \leftrightarrow ((Smoke \rightarrow Fire) \vee (Heat \rightarrow Fire))$
6. $(Smoke \rightarrow Fire) \rightarrow ((Smoke \wedge Heat) \rightarrow Fire)$
7. $Big \vee Dumb \vee (Big \rightarrow Dumb)$

Solution 3.

1. valid.
2. satisfiable, not valid.
3. satisfiable, not valid.
4. valid.
5. valid! This one is interesting. Let's reason.
   (a) Suppose the right hand side is true. We will show the LHS is also true. To show an implication is true, suppose the antecedent holds, i.e., both *Smoke* and *Heat* are true. We want to show *Fire* is true. Since the disjunction is true, at least one of the two disjuncts is true. Say it is $Smoke \rightarrow Fire$. But since we know *Smoke* is true, also *Fire* is true, which is what we wanted to show. The other case, that $Heat \rightarrow Fire$ were true is similar.

(b) Suppose the left hand side is true. We will show the RHS is also true. To do this, we assume by contradiction that it is not.The only way to make a disjunction false is if both disjuncts are false. So we must have that *Smoke* is true, *Fire* is false, and *Heat* is true. But then the left hand side is not true, a contradiction.

6. valid

7. valid.

Problem 4. Suppose you had code that solved the satisfiability problem. Show how you could call that code to solve the validity problem, i.e., to test if a given formula is valid or not.

Solution 4. Here is how to solve the validity problem if you had code for the satisfiability problem: take the input formula F, negate it to get $\neg F$, and pass it to the satisfiability checker. If you get the result that $\neg F$ is satisfiable, then you know that F is not valid; and if you get the result that $\neg F$ is not satisfiable, then you know that F is valid. Do you see why?

Problem 5.[Assignment problem in 2021] Each of the following statements about propositional logic is false. For each statement, show why the statement is false with a counterexample using only the variables p, q (you don't need to use both variables).

1. If F is valid and G is satisfiable then $(F \to G)$ is valid.
2. If F is satisfiable and G is satisfiable then $(F \wedge G)$ is satisfiable.
3. If F is valid then $\neg F$ is satisfiable.
4. If F is satisfiable then $\neg F$ is satisfiable.

Solution 5. We will use the following formulas:

- $p \vee \neg p$ which is valid since if an assignment makes F false then either it makes p equal to 0 and it makes $\neg p$ equal to 0, which is impossible.
- q which is satisfiable, by the assignment that maps q to 1.
- $\top$ which is valid (by definition of the semantics of $\top$).

(a) Take $F = p \vee \neg p$ and $G = q$. Then $F \to G = (p \vee \neg p) \to q$ is not valid, e.g., take the assignment that maps p to 0 and q to 0.

(b) Take $F = p$ and $G = \neg p$. Then $F \wedge G = p \wedge \neg p$ is not satisfiable since no assignment can map p to 1 and $\neg p$ to 1.

(c) Take $F = \top$. Then F is valid but $\neg F = \neg\top \equiv \bot$ is not satisfiable (by definition of the semantics of $\bot$).

(d) Take $F = \top$. Then F is satisfiable but $\neg F = \neg\top \equiv \bot$ is not satisfiable (by definition of the semantics of $\bot$).

Problem 6. Write a recursive function **SubForms**(G) that returns the set of subformulas of G (i.e. in a similar style to the way that the truth value function tv is defined in the lecture). Do this for the basic syntax, and for the extended syntax.

Solution 6. The set of subformulas of a formula G is denoted **SubForms**(G), and is defined as follows:

1. If G is an atom, then **SubForms**$(G) = \{G\}$.
2. If G is a formula of the form $\neg F$, then **SubForms**$(G) = \{G\} \cup$ **SubForms**(F).
3. If G is a formula of the form $(E \vee F)$ or $(E \wedge F)$ then **SubForms**$(G) = \{G\} \cup$ **SubForms**$(E) \cup$ **SubForms**(F).

To handle $\rightarrow, \leftrightarrow, \top, \bot$ use the following rules:

4. If G is a formula of the form $\top$ or $\bot$, then **SubForms**$(G) = \{G\}$.
5. If G is a formula of the form $(E \rightarrow F)$ or $(E \leftrightarrow F)$, then **SubForms**$(G) = \{G\} \cup$ **SubForms**$(E) \cup$ **SubForms**(F).

You may also want to handle $\bigvee, \bigwedge$:

6. If G is a formula of the form $(\bigwedge_{i=1}^{n} F_i)$ or $(\bigvee_{i=1}^{n} F_i)$ then **SubForms**$(G) = \{G\} \cup$ **SubForms**$(F_1) \cup \cdots \cup$ **SubForms**(F_n).

This illustrates the usefuleness of having a definition with just a few connectives, i.e., there is less to write.

Problem 7. Intuitively, the size of a propositional formula F is the number of symbols in F, ignoring parentheses. So, e.g., $|(p \vee \neg p)| = 4$.

1. Give a recursive procedure defining the size of F, denoted $|F|$.
2. Show by induction that the number of subformulas of F, denoted $|$**SubForms**$(F)|$, is at most $|F|$. Hint: use induction on the size of F.

Solution 7.

1. The size of a formula $|F|$ is defined using the following recursive procedure:
   (a) If q is an atom, then $|q|$ is 1.
   (b) The size of a formula of the form $\neg F$ is $|F| + 1$.

(c) The size of a formula of the form $(F \vee G)$ is $|F| + |G| + 1$.

(d) The size of a formula of the form $(F \wedge G)$ is $|F| + |G| + 1$.

2. We induct on the size of F to show that $|\mathbf{SubForms}(F)| \leq |F|$.

   (a) **Base case:** If F has size 1, it is an atom, say q. Then $|\mathbf{SubForms}(q)|$

   $$\begin{aligned} &= |\{q\}| && \text{(dfn of subformula)} \\ &= 1 \\ &= |q| && \text{(dfn of formula size)} \end{aligned}$$

   (b) **Inductive cases:** Suppose F has size > 1. Then either $F = \neg G$, or $F = G \vee H$, or $F = G \wedge H$ for some formulas G, H of size smaller than F. (So, by induction, we know that $|\mathbf{SubForms}(G)| \leq |G|$ and $|\mathbf{SubForms}(H)| \leq |H|$, facts that we will use shortly).
   We treat each case separately:

      i. If $F = \neg G$ then $|\mathbf{SubForms}(\neg G)|$

      $$\begin{aligned} &= |\{\neg G\} \cup \mathbf{SubForms}(G)| && \text{(dfn of subformula)} \\ &= 1 + |\mathbf{SubForms}(G)| \\ &\leq 1 + |G| && \text{(by induction)} \\ &= |\neg G| && \text{(by definition of formula size)} \end{aligned}$$

      ii. If $F = G \vee H$ then $|\mathbf{SubForms}(G \vee H)|$

      $$\begin{aligned} &= |\{G \vee H\} \cup \mathbf{SubForms}(G) \cup \mathbf{SubForms}(H)| && \text{(dfn of subformula)} \\ &= 1 + |\mathbf{SubForms}(G)| + |\mathbf{SubForms}(H)| \\ &\leq 1 + |G| + |H| && \text{(induction)} \\ &= |G \vee H| && \text{(dfn of formula size)} \end{aligned}$$

      iii. If $F = G \wedge H$ then the argument is symmetric to the previous case, just replace $\vee$ by $\wedge$.

   This completes the proof.

In this proof we inducted on the size of the formula. But all this really did was break our argument into the base case (which we had to prove directly), and the three inductive cases in which F is build from other formulas G, H (where we made use of the fact that the claim holds on G, H). Since this is all that is really needed to make this proof work, we can dispense with the beginning of the proof that talks about induction on the size of the formula. Instead, we can say we are inducting on the *structure* of the formula. This is called *structural induction*. Here is the same proof, this time by structural induction, and without the justifications inside each case (since they are identical to the ones before).

1. **Base case (F is an atom):** If F is an atom, then $|\mathbf{SubForms}(F)| = 1 = |F|$.

2. **Inductive cases (F is of the form (a) $\neg G$, (b) $G \vee H$, and (c) $G \wedge H$):**

(a) $|\mathbf{SubForms}(\neg G)| = |\{\neg G\} \cup \mathbf{SubForms}(G)| = 1 + |\mathbf{SubForms}(G)| \leq 1 + |G| = |\neg G|$

(b) $|\mathbf{SubForms}(G \vee H)| = |\{G \vee H\} \cup \mathbf{SubForms}(G) \cup \mathbf{SubForms}(H)| = 1 + |\mathbf{SubForms}(F)| + |\mathbf{SubForms}(G)| \leq 1 + |F| + |G| = |F \vee G|$.

(c) $|\mathbf{SubForms}(G \wedge H)| = |\{G \wedge H\} \cup \mathbf{SubForms}(G) \cup \mathbf{SubForms}(H)| = 1 + |\mathbf{SubForms}(F)| + |\mathbf{SubForms}(G)| \leq 1 + |F| + |G| = |F \wedge G|$.