Validity without Truth Values

Let me first show that we do not need truth values in order to define the all-important notions of validity and satisfiability.

Definition 1 *Let S be a set of formulas. The set S is a* truth set *(also called saturated) if the following conditions hold:*

(i) For all pairs of formulas $(\varphi, \neg\varphi)$, exactly one of φ or $\neg\varphi$ is in S;

(ii) If $\varphi \wedge \psi$ is in S, then φ and ψ are both in S;

(iii) If $\varphi \vee \psi$ is in S, then φ or ψ is in S;

(iv) If $\varphi \Rightarrow \psi$ is in S, then $\neg\varphi$ or ψ is in S.

Since there are infinitely many formulas, condition (i) ensures that every truth set has infinitely many formulas.

For any set of formulas S, we can define a valuation v_S by taking $v_S(\varphi) = t$ if $\varphi \in S$.

Proposition 2 *Let S be a set of formulas. The following are equivalent:*

(i) S is a truth set;

(ii) v_S is a Boolean valuation.

Proof. By induction on formulas. □

Similarly, if v is a valuation, we can define a set of formulas $S_v = \{\varphi \mid v(\varphi) = t\}$.

Proposition 3 *Let v be a valuation. The following are equivalent:*

(i) v is a Boolean valuation;

(ii) S_v *is a truth set.*

Proof. By induction on formulas. □

Thus, we can go back and forth between truth sets and Boolean valuations. We can now define validity and satisfiability using only truth sets.

Proposition 4 *Let* $\mathcal{S}$ *be the set of all truth sets. The formula* φ *is valid if and only if*

$$\varphi \in \bigcap_{S \in \mathcal{S}} S.$$

The formula φ *is satisfiable if and only if*

$$\varphi \in \bigcup_{S \in \mathcal{S}} S.$$

Proof. Immediate by the definitions. □

Propositional Logic with Constants

An important variant of propositional logic is one where we introduce *constants* $true$ and $false$ to stand for (unsurprisingly) the truth values t and f. We call this variant PLC (for propositional logic with constants).

More precisely, we define formulas as we have for propositional logic, except that we allow $true$ and $false$ as additional atomic formulas. Note that $true$ and $false$ are *syntax*.

To give truth values to PLC formulas, we extend the definition of a Boolean valuation to specify the truth values to assign with $true$ and $false$. A Boolean valuation v is required to assign:

$$v(true) = t$$
$$v(false) = f.$$

All the other definitions apply in the obvious way. To distinguish between truth for PL and for PLC, we write $v_0 \models_c \varphi$ to emphasize that the formula φ of PLC is true under interpretation v_0. Similarly, we write $\models_c \varphi$ if a formula φ of PLCS is true under all interpretations.

There is of course a relationship between validity for in PL and validity in PLC. Roughly speaking, by adding constants $true$ and $false$, we do not "kill" any tautologies that exist in PL. Formally, we have the following *conservative extension* result.

Proposition 5 *Let φ be a formula of PL. Because every formula of PL is also a syntactically well-formed formula of PLC, we can also view φ as a formula of PLC. We have $\models \varphi$ if and only if $\models_c \varphi$.*

Proof. By induction on formulas of PL. □

Of course, the result does not hold for all formulas of PLC, since there are formulas of PLC (the ones that use $true$ and $false$) that are not formulas of PL.

Here are some examples of valid formulas of PLC:

(1) $\models_c \neg false \Leftrightarrow true$

(2) $\models_c (\varphi \wedge true) \Leftrightarrow \varphi$

(3) $\models_c (\varphi \wedge false) \Leftrightarrow false$

(4) $\models_c (\varphi \Rightarrow false) \Leftrightarrow \neg\varphi$

Statement (4) says that $\varphi \Rightarrow false$ and $\neg\varphi$ are logically equivalent. This means that if we have the constant $false$, we can express negation using implication. Following what we already know about PL, along with (1), this means that we can derive all of PLC using only, say, implication and $false$.

Boolean Decision Diagrams

One reason for introducing PLC is to talk about the following. For a propositional variable p, define the formula $p \rightarrow \varphi, \psi$ as an abbreviation for $(p \wedge \varphi) \vee (\neg p \wedge \psi)$. Intuitively, this is an if-then-else statement: to establish the truth value of $p \rightarrow \varphi, \psi$, check the truth value of p: if it is true, the truth value of $p \rightarrow \varphi, \psi$ is given by the truth value of φ; otherwise, it is given by the truth value of ψ. (Why?)

Note, for instance, that $\neg p$ can be written (is logically equivalent to) $p \rightarrow falsep, true$. Similarly, $p \wedge q$ can be written $p \rightarrow (q \rightarrow true, false), (q \rightarrow false, false)$ or, equivalently, $p \rightarrow q, false$.

Let $\varphi[x \mapsto \psi]$ represent the formula φ where every occurrence of x is replaced by the formula ψ.

Proposition 6 *Let φ be a formula of PLC, and let p be a propositional variable. The formula φ is logically equivalent to $p \rightarrow \varphi[p \mapsto true], \varphi[p \mapsto false]$. The latter is called the* Shannon expansion of φ with respect to p.

Proof. By induction on formulas. □

We define a normal form sometimes called INF (for If-then-else Normal Form) as a formula built entirely from the if-then-else operator and the constants $true$ and $false$. We can construct an INF as follows: if φ contains no propositional variables, it is either logically equivalent to $true$ or to $false$, which is an INF. Otherwise, we form the Shannon expansion of φ with respect to one of the variables p in φ. Since $\varphi[p \mapsto false]$ and $\varphi[p \mapsto true]$ both contain one less variable than φ, we can apply this procedure recursively to find INFs for both of these (say, φ_1, φ_2), from which we can construct the INF for φ as $p \to \varphi_1, \varphi_2$. This essentially proves:

Proposition 7 *Any formula of PLC is logically equivalent to a formula in INF.*

Now, the process of constructing an INF can be visualized as a directed acyclic graph from which we can read off the truth value of a formula under any interpretation. Consider the formula $(p_1 \vee q_1) \wedge (p_2 \vee q_2)$. Let's apply the above construction recursively, by expanding successively on the variables p_1, q_1, p_2, q_2, and by naming all the intermediate formulas we obtain. (I write $\psi \rightsquigarrow \psi'$ for a transformation of a formula into a logically equivalent formula.) I have boxed the INF at each step.

$$\varphi = (p_1 \vee q_1) \wedge (p_2 \vee q_2) \rightsquigarrow \boxed{p_1 \to \varphi_t, \varphi_f}$$

$$\varphi_t = (true \vee q_1) \wedge (p_2 \vee q_2) \rightsquigarrow p_2 \vee q_2 \rightsquigarrow \boxed{q_1 \to \varphi_{tt}, \varphi_{tf}}$$

$$\varphi_{tt} = p_2 \vee q_2 \rightsquigarrow \boxed{p_2 \to \varphi_{ttt}, \varphi_{ttf}}$$

$$\varphi_{ttt} = true \vee q_2 \rightsquigarrow \boxed{true}$$

$$\varphi_{ttf} = false \vee q_2 \rightsquigarrow q_2 \rightsquigarrow \boxed{q_2 \to true, false}$$

$$\varphi_{tf} = p_2 \vee q_2 \rightsquigarrow \boxed{p_2 \to \varphi_{tft}, \varphi_{tff}}$$

$$\varphi_{tft} = true \vee q_2 \rightsquigarrow \boxed{true}$$

$$\varphi_{tff} = false \vee q_2 \rightsquigarrow q_2 \rightsquigarrow \boxed{q_2 \to true, false}$$

$$\varphi_f = (false \vee q_1) \wedge (p_2 \vee q_2) \rightsquigarrow q_1 \wedge (p_2 \vee q_2) \rightsquigarrow \boxed{q_1 \to \varphi_{ft}, \varphi_{ff}}$$

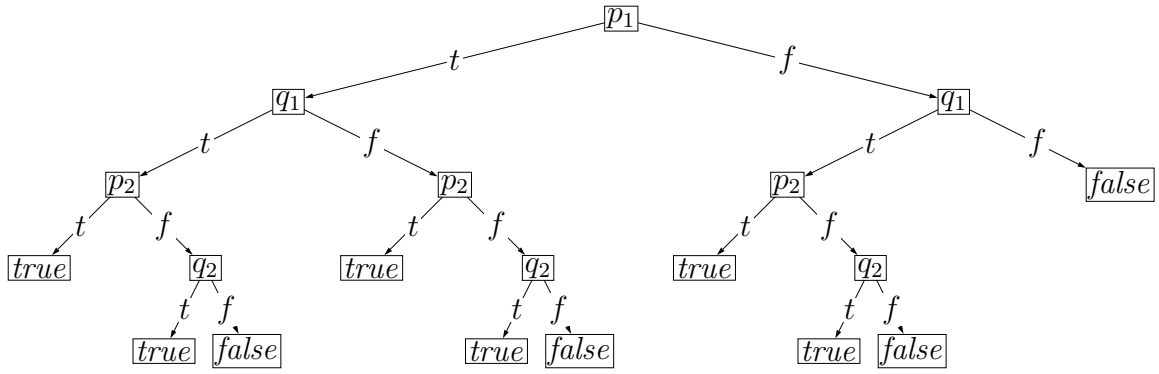
$$\varphi_{ft} = true \wedge (p_2 \vee q_2) \rightsquigarrow p_2 \vee q_2 \rightsquigarrow \boxed{p_2 \to \varphi_{ftt}, \varphi_{ftf}}$$

$$\varphi_{ftt} = true \vee q_2 \rightsquigarrow \boxed{true}$$

$$\varphi_{ftf} = false \vee q_2 \rightsquigarrow q_2 \rightsquigarrow \boxed{q_2 \to true, false}$$

$$\varphi_{ff} = false \wedge (p_2 \vee q_2) \rightsquigarrow \boxed{false}$$

We can draw this as a tree, where the nodes represent the formulas at every step. I've labeled the nodes by the variable on which the formula is expanded. There are two children for every node, one representing the formula obtained when the variable is true, and one when the variable is false.



We can now read off the truth value of a formula under any interpretation by starting from the root of the tree, and following either the branch marked t or f, depending on the truth value given by the interpretation to the variable at the node. Moreover, a formula is a tautology if the leaves all contain $true$, and a formula is satisfiable if at least one leaf contains $true$.

You might have noticed that we don't actually need to construct the whole tree, but we can notice that some formulas are repeated in the construction. For instance, φ_{tt} and φ_{tf} are the same formulas, since after all, $p_2 \vee q_2$ does not mention q_1, so the Shannon expansion with respect to q_1 does not change the formula. Therefore, we can collapse the two branches of the tree, and avoid redoing the work. If we do this consistently, then we end up with an acyclic directed graph. (It is of course possible to do this directly when generated the INF form; it's a good exercise to figure out how.)