Propositional Logic: Syntax and Semantics

Mahesh Viswanathan

Fall 2018

Modern logic is a formal, symbolic system that tries to capture the principles of correct reasoning and truth. To describe any formal language precisely, we need three pieces of information — the *alphabet* describes the symbols used to write down the sentences in the language; the *syntax* describes the rules that must be followed for describing "grammatically correct" sentences in the language; and finally, the *semantics* gives "meaning" to the sentences in our formal language. Most of you have already encountered other contexts where formal languages were introduced in such a manner. Here are some illustrative examples.

Example 1. Binomial coefficients are written using natural numbers and parentheses. However, not every way to put together parenthesis and natural numbers is a binomial coefficient. For example, $(1, (1)$, or $(_2)$ are examples of things that are no binomial coefficients. Correctly formed bionomial coefficients are of the form $\binom{i+j}{i}$, where i and j are natural numbers. We could define the meaning of $\binom{i+j}{i}$ to be the natural number $\frac{(i+j)!}{i!j!}$. On the other hand, we could define the meaning of $\binom{i+j}{i}$ to be the number of ways of choosing i elements from a set of $i+j$ elements. Though both these ways of interpreting binomial coefficients are the same, they have a very different presentation. In general, one could define semantics in different ways, or even very different semantics to the same syntactic objects.

Example 2. Precise definitions of programming languages often involve characterizing its syntax and semantics. Here is an extremely simple programming language drawing called Turtle. Programs in this language are written using $\mathbf{F}$, $+$, and $-$. Any sequence formed by such symbols is a syntactically correct program in this language. We will interpret such a sequence of symbols as instructions to draw a picture — $\mathbf{F}$ is an instruction to draw a line by moving forward 1 unit; $+$ is an instruction to turn the heading direction $60°$ to the left; $-$ is an instruction to turn the heading direction $60°$ to the right. Figure 1 shows example programs and the pictures they draw based on this interpretation.

Even though the Turtle language is a very simple programming language, some very interesting curves can be approximated. Consider the following iterative procedure that produces a sequence of programs. Start with the program $\mathbf{F}$. In each iteration, if P is a program at the start of the iteration, then construct the program P' obtained by replacing each $\mathbf{F}$ in P by $\mathbf{F}+\mathbf{F}--\mathbf{F}+\mathbf{F}$. So at the beginning we have program $\mathbf{F}$, in the next iteration the program is $\mathbf{F}+\mathbf{F}--\mathbf{F}+\mathbf{F}$, and in the iteration after that it will be

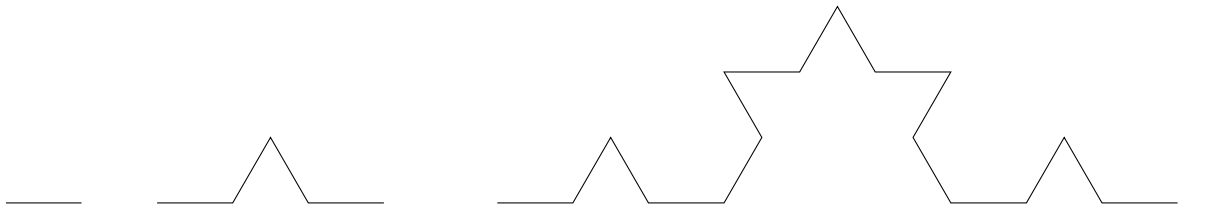

Figure 1: Example Turtle programs and the pictures they draw.

$\mathbf{F}+\mathbf{F}--\mathbf{F}+\mathbf{F}+\mathbf{F}+\mathbf{F}--\mathbf{F}+\mathbf{F}--\mathbf{F}+\mathbf{F}--\mathbf{F}+\mathbf{F}+\mathbf{F}+\mathbf{F}--\mathbf{F}+\mathbf{F}$, and so on. The programs in this sequence draw pictures that in the limit approach the Koch curve.

Example 3. Regular expressions are expressions that define special collections of strings over some alphabet called regular languages. Regular expressions over an alphabet Σ are built up using Σ, parentheses, $\emptyset$, ε, $\cdot$, $+$, and *. Inductively, they are defined as the smallest set that satisfy the following rules.

- $\emptyset$ and ε are regular expressions.
- For any $a \in \Sigma$, a is a regular expression.
- If r_1, r_2 are regular expressions then so are $(r_1 \cdot r_2)$, $(r_1 + r_2)$, and (r_1^*).

Each regular expression r, semantically defines a subset of Σ^* [1] that we will denote by $[\![r]\!]$. The semantics of regular expressions is defined inductively as follows.

- $[\![\emptyset]\!] = \emptyset$, and $[\![\varepsilon]\!] = \{\varepsilon\}$.
- For $a \in \Sigma$, $[\![a]\!] = \{a\}$.
- Inductively, $[\![(r_1 + r_2)]\!] = [\![r_1]\!] \cup [\![r_2]\!]$, $[\![(r_1 \cdot r_2)]\!] = [\![r_1]\!] \cdot [\![r_2]\!]$ and $[\![(r_1^*)]\!] = [\![r_1]\!]^*$, where $\cdot$ (on the right hand side) denotes the concatenation of two languages, and * denotes the Kleene closure of a language.

We will now define one of the simplest logics one encounters in an introductory discrete mathematics class. It is called *propositional* or *sentential* logic. This logic is a symbolic language to reason about *propositions*. Propositions are declarative sentences that are either true or false. Examples of such include "Springfield is the capital of Illinois", "1+1 = 2", "2+2 = 3". Notice that propositions don't need to be true facts (like "2+2 = 3"), but they must be something is either true or false. Statements that are **not** propositions are things like "What is it?", "Location of robot", "x+1 = 2". The logic itself will be symbolic and abstract away from english sentences like the ones above. We will introduce a precise definition of this logic, much in the same way as Example 3, defining the syntax and semantics inductively.

1 Syntax

To define this logic, we will assume a (countably infinite) set of *propositions* $\mathsf{Prop} = \{p_i \,|\, i \in \mathbb{N}\}$. The formulas of propositional logic will be strings over the alphabet $\mathsf{Prop} \cup \{(,), \to, \bot\}$; here $\to$ is called *implication*, and $\bot$ is called *false*.

Definition 4. The set of *well formed formulas* (wff) in propositional logic is the smallest set satisfying the following properties.

- $\bot$ is a wff.
- Any proposition p_i (by itself) is a wff.
- If φ and ψ are wffs then $(\varphi \to \psi)$ is a wff.

Examples of wffs include p_1, $\bot$, $(p_1 \to p_2)$, $(((p_1 \to p_3) \to (p_1 \to p_4)) \to p_1)$. On the other hand the following strings are not wffs: $p_1 \to$, $\to \bot$, $\bot p_1$.

Inductive definitions of the kind in Example 3 or Definition 4 are quite common when defining the syntax of formulas in a logic or of programming languages. Therefore, in computer science, one often uses a "grammar-like" presentation for the syntax. For example, wffs φ in propositional logic are given by the following *BNF grammar*.

$$\varphi ::= p \mid \bot \mid (\varphi \to \varphi)$$

[1] For a finite set Σ, Σ^* denotes the collection of (finite) sequences/strings/words over Σ. For $n \in \mathbb{N}$, we use Σ^n to denote the set of sequences/strings/words over Σ of length exactly n.

where p is an element of Prop. Reading such grammars takes some getting used to. For example, the rule $(\varphi \to \varphi)$ doesn't mean that implications can only be used when the two arguments are the same. Instead it says that if we take two elements that belong to the syntactic entity φ (i.e., wffs), put $\to$ between them with surrounding parenthesis, then we get another element belonging to the same syntactic entity as φ. We will sometimes use such a grammar representation to describe syntax in a succinct manner.

Other logical operators. The syntax for propositional logic presented above doesn't have the usual Boolean operations of negation, disjunction, and conjunction. However, these can be conveniently defined in terms of the operators we have used. We will consider these standard Boolean operations as "derived logical operations" as follows — for wffs φ and ψ, $(\neg\varphi)$ denotes the formula $(\varphi \to \bot)$, $(\varphi \vee \psi)$ denotes the formula $((\neg\varphi) \to \psi)$, and $(\varphi \wedge \psi)$ denotes the formula $(\neg((\neg\varphi) \vee (\neg\psi)))$. Another useful wff is $\top$ (read as "true"); $\top$ denotes the formula $(\neg\bot)$ or $(\bot \to \bot)$.

Notation. Writing formulas strictly according to the syntax presented will become cumbersome because of many parentheses and subscripts. Therefore, we will make the following notational simplifications.

- The outermost parentheses will be dropped. Thus we will write $p_3 \to (\bot \to p_1)$ instead of $(p_3 \to (\bot \to p_1))$
- We will sometimes omit subscripts of propositions. Thus we will write p instead of p_1, or q instead of p_2, r instead of p_3, or s instead of p_4, and so on.
- The following precedence of operators will be assumed: $\neg$, $\wedge$, $\vee$, $\to$. Thus $\neg p \wedge q \to r$ will mean $(((\neg p) \wedge q) \to r)$

Our inductive definition of wffs in propositional logic has the nice property that the structure of a formula can be interpreted in a unique way. There is no ambiguity in its interpretation. For example, if $p_1 \to p_2 \to p_3$ were a wff, then it is unclear whether we mean the formula $\varphi = ((p_1 \to p_2) \to p_3)$ or $\psi = (p_1 \to (p_2 \to p_3))$ — in φ $(p_1 \to p_2)$ and p_3 are the arguments to the topmost $\to$, while in ψ p_1 and $(p_2 \to p_3)$ are the arguments to the topmost $\to$. Our syntax does not have such issues. This will be exploited often in inductive definitions and in algorithms. This observation can be proved by structural induction, but we skip its proof.

Theorem 5 (Unique Readability)**.** *Any wff can be uniquely read, i.e., it has a unique topmost logical operator and well defined immediate sub-formulas.*

2 Semantics

We will now provide a meaning or *semantics* to the formulas. Our definition will following the inductive definition of the syntax, just like in Example 3. The semantics of formulas in a logic, are typically defined with respect to a *model*, which identifies a "world" in which certain facts are true. In the case of propositional logic, this world or model is a *truth valuation* or *assignment* that assigns a truth value (true/false) to every proposition. The *truth value truth* will be denoted by 1, and the truth value *falsity* will be denoted by 0.

Definition 6. A *(truth) valuation* or *assignment* is a function v that assigns truth values to each of the propositions, i.e., $\mathsf{v} : \mathsf{Prop} \to \{0, 1\}$.

The value of a proposition p under valuation v is given by $\mathsf{v}(p)$.

We will define the semantics through a *satisfaction relation*, which is a binary relation $\models$ between valuations and formulas. The statement $\mathsf{v} \models \varphi$ should be read as "v satisfies φ" or "φ is true in v" or "v is a model of φ". It is defined inductively following the syntax of formulas. In the definition below, we say $\mathsf{v} \not\models \varphi$ when $\mathsf{v} \models \varphi$ does not hold.

Definition 7. For a valuation v and wff φ, the satisfaction relation, $\mathsf{v} \models \varphi$, is defined inductively based on the structure of φ as follows.

- $\mathsf{v} \models \bot$ is *never true*. That is, $\mathsf{v} \not\models \bot$.
- $\mathsf{v} \models p$ if $\mathsf{v}(p) = 1$.
- $\mathsf{v} \models (\varphi \to \psi)$ if either $\mathsf{v} \models \psi$ or $\mathsf{v} \not\models \varphi$.

Example 8. Let us look at a couple of examples to see how the inductive definition of the satisfaction relation can be applied. Consider the formula $\varphi = (p \to (q \to r)) \to ((p \to q) \to (p \to r))$. Consider the valuation v_1 that sets all propositions to 1. Now $\mathsf{v}_1 \models \varphi$ can be seen from the following observations.

$\mathsf{v}_1 \models r$	because $\mathsf{v}_1(r) = 1$
$\mathsf{v}_1 \models p \to r$	semantics of $\to$
$\mathsf{v}_1 \models (p \to q) \to (p \to r)$	semantics of $\to$
$\mathsf{v}_1 \models (p \to (q \to r)) \to ((p \to q) \to (p \to r))$	semantics of $\to$

Consider v_2 that assigns all propositions to 0. Once again $\mathsf{v}_2 \models \varphi$. The reasoning behind this observation is as follows.

$\mathsf{v}_1 \not\models p$	because $\mathsf{v}_1(p) = 0$
$\mathsf{v}_1 \models p \to r$	semantics of $\to$
$\mathsf{v}_1 \models (p \to q) \to (p \to r)$	semantics of $\to$
$\mathsf{v}_1 \models (p \to (q \to r)) \to ((p \to q) \to (p \to r))$	semantics of $\to$

The semantics in Definition 7 defines a satisfaction relation between valuations and formulas. However, one could defined the semantics of propositional logic differently, by considering the formula as a "program" or "circuit" that computes a truth value based on the assignment. This approach is captured by the following definition of the *value* of a wff under a valuation.

Definition 9. The *value of a wff* φ *under valuation* v, denoted by $\mathsf{v}[\![\varphi]\!]$, is inductively defined as follows.

$$\begin{aligned}
&\mathsf{v}[\![\bot]\!] = 0 \\
&\mathsf{v}[\![p]\!] = \mathsf{v}(p) \\
&\mathsf{v}[\![\varphi \to \psi]\!] = \begin{cases} 0 & \text{if } \mathsf{v}[\![\varphi]\!] = 1 \text{ and } \mathsf{v}[\![\psi]\!] = 0 \\ 1 & \text{otherwise} \end{cases}
\end{aligned}$$

Example 10. Let us consider $\varphi = (p \to (q \to r)) \to ((p \to q) \to (p \to r))$ and v_1 which assigns all propositions to 1, from Example 8. $\mathsf{v}_1[\![\varphi]\!]$ can be computed as follows.

$\mathsf{v}_1[\![r]\!] = 1$	because $\mathsf{v}_1(r) = 1$
$\mathsf{v}_1[\![p \to r]\!] = 1$	semantics of $\to$
$\mathsf{v}_1[\![(p \to q) \to (p \to r)]\!] = 1$	semantics of $\to$
$\mathsf{v}_1[\![(p \to (q \to r)) \to ((p \to q) \to (p \to r))]\!] = 1$	semantics of $\to$

Definitions 7 and 9 are both equivalent in some sense. This is captured by the following theorem.

Theorem 11. *For any truth valuation* v *and wff* φ, $\mathsf{v} \models \varphi$ *if and only if* $\mathsf{v}[\![\varphi]\!] = 1$

The proof of Theorem 11 is by structural induction on the formula φ. It is straightforward and is left as an exercise for the reader.

It is convenient to associate with every wff a set of truth valuations. These are the valuations under which the formula holds.

Definition 12. The *models* of wff φ is the set of valuations that *satisfy* φ. More precisely,

$$[\![\varphi]\!] = \{\mathsf{v} \mid \mathsf{v} \models \varphi\}.$$

Observe that as per the definition, $[\![\bot]\!] = \emptyset$.

Truth valuations as defined are an "infinite object" since they assign a truth value to all propositions, and we have a countably infinite set of propositions. However, for a fixed formula φ, only the truth values assigned to the finitely many propositions syntactically appearing in φ matter. This is captured in our next theorem. However, before presenting it, let us inductively define the propositions that appear in a formula. For a wff φ, the set of propositions appearing in φ, denoted $\mathsf{prop}(\varphi)$, is inductively defined as follows.

$$\begin{aligned}\mathsf{prop}(\bot) &= \emptyset\\ \mathsf{prop}(p) &= \{p\}\\ \mathsf{prop}((\varphi \to \psi)) &= \mathsf{prop}(\varphi) \cup \mathsf{prop}(\psi)\end{aligned}$$

Theorem 13. *Let v_1 and v_2 be truth valuations such that for all $p \in \mathsf{prop}(\varphi)$, we have $\mathsf{v}_1(p) = \mathsf{v}_2(p)$, i.e., v_1 and v_2 agree on the truth values assigned to all propositions in $\mathsf{prop}(\varphi)$. Then $\mathsf{v}_1 \models \varphi$ if and only if $\mathsf{v}_2 \models \varphi$.*

Proof. By structural induction on φ.

Base Case $\varphi = \bot$ By definition $\mathsf{v}_1 \not\models \varphi$ and $\mathsf{v}_2 \not\models \varphi$.

Base Case $\varphi = p$ Observe that, $\mathsf{v}_1 \models \varphi$ iff $\mathsf{v}_1(p) = 1 = \mathsf{v}_2(p)$ iff $\mathsf{v}_2 \models \varphi$.

Induction Step Consider $\varphi = (\psi_1 \to \psi_2)$. Since $\mathsf{prop}(\psi_i) \subseteq \mathsf{prop}(\varphi)$ (for $i \in \{1, 2\}$), we have by induction hypothesis, $\mathsf{v}_1 \models \psi_i$ iff $\mathsf{v}_2 \models \psi_i$. Therefore, by definition of the semantics of $\to$, $\mathsf{v}_1 \models \varphi$ iff $\mathsf{v}_2 \models \varphi$.

□

The main consequence Theorem 13 is that, to determine if a formula holds in a model, we only need to consider the assignment to finitely many propositions. Thus, instead of thinking of valuations as assigning truth values to all propositions, we can think of them as only assigning values to the propositions of interest, which will be clear from the context. So we will typically think of valuations as functions with a finite domain.

3 Satisfiability and Validity

Two formulas that are syntactically different, could however, be "semantically equivalent". But what do we mean by semantic equivalence? Intuitively, this is when the truth value of each formula in every valuation is the same.

Definition 14 (Logical Equivalence)**.** A wff φ is said to be *logically equivalent* to ψ iff any of the following equivalent conditions hold.

- for every valuation v, $\mathsf{v} \models \varphi$ iff $\mathsf{v} \models \psi$,
- for every valuation v, $\mathsf{v}[\![\varphi]\!] = \mathsf{v}[\![\psi]\!]$,
- $[\![\varphi]\!] = [\![\psi]\!]$.

We denote this by $\varphi \equiv \psi$.

Let us consider an example to see how we may reason about two formulas being logically equivalent.

Example 15. Consider the wffs $\varphi_1 = p \to q$ and $\varphi_2 = \neg q \to \neg p$, where p and q are propositions. Though φ_1 and φ_2 are syntactically different, they are semantically equivalent. To prove that $\varphi_1 \equiv \varphi_2$, we need to show that they two formulas evaluate to the same truth value under every valuation. One convenient way to oragnize such a proof is as *truth table*, where different cases in the case-by-case analysis correspond to different rows. Each row of the truth table corresponds to a (infinite) collection of valuations based on the value assigned to propositions p and q; the columns correspond to the value of different (sub)-formulas under each valuation in this collection. For example, a truth table reasoning for φ_1 and φ_2 will look as follows.

$\mathsf{v}[\![p]\!]$	$\mathsf{v}[\![q]\!]$	$\mathsf{v}[\![p \to q]\!]$	$\mathsf{v}[\![\neg p]\!]$	$\mathsf{v}[\![\neg q]\!]$	$\mathsf{v}[\![\neg q \to \neg p]\!]$
1	1	1	0	0	1
1	0	0	0	1	0
0	1	1	1	0	1
0	0	1	1	1	1

Notice that since the third column and the last (sixth) column are identical for every row, and every valuation corresponds to some row in the table, it follows that φ_1 and φ_2 are logically equivalent.

Let us now consider $\varphi_1' = \psi_1 \to \psi_2$ and $\varphi_2' = \neg\psi_2 \to \neg\psi_1$, where ψ_1 and ψ_2 are arbitrary formulas. Once again φ_1' and φ_2' are logically equivalent, and the reasoning is essentially the same as above. The rows of the truth table now classify valuations based on the value of formulas ψ_1 and ψ_2 under them.

$\mathsf{v}[\![\psi_1]\!]$	$\mathsf{v}[\![\psi_2]\!]$	$\mathsf{v}[\![\psi_1 \to \psi_2]\!]$	$\mathsf{v}[\![\neg\psi_1]\!]$	$\mathsf{v}[\![\neg\psi_2]\!]$	$\mathsf{v}[\![\neg\psi_2 \to \neg\psi_1]\!]$
1	1	1	0	0	1
1	0	0	0	1	0
0	1	1	1	0	1
0	0	1	1	1	1

Truth table based reasoning, as carried out in Example 15, is a very convenient way to organize proofs of propositional logic. We will often use it.

Definition 16 (Logical Consequence)**.** Let Γ be a (possibly infinite) set of formulas and let φ be a wff. We say that φ is a *logical consequence* of Γ (denoted $\Gamma \models \varphi$) iff for every valuation v, if for every $\psi \in \Gamma$, $\mathsf{v} \models \psi$ then $\mathsf{v} \models \varphi$. In other words, any model that satisfies every formula in Γ also satisfies φ.

We could equivalently have defined it as $\Gamma \models \varphi$ iff $\bigcap_{\psi\in\Gamma}[\![\psi]\!] \subseteq [\![\varphi]\!]$.

Example 17. Consider the set $\Gamma = \{\psi_1 \to \psi_2, \psi_2 \to \psi_1, \psi_1 \vee \psi_2\}$, where ψ_1 and ψ_2 are arbitrary formulas. We will show that $\Gamma \models \psi_1$. Once again, we will use a truth table to classify valuations into row based on the value that ψ_1 and ψ_2 evaluate to. Such a truth table looks as follows.

$\mathsf{v}[\![\psi_1]\!]$	$\mathsf{v}[\![\psi_2]\!]$	$\mathsf{v}[\![\psi_1 \to \psi_2]\!]$	$\mathsf{v}[\![\psi_2 \to \psi_1]\!]$	$\mathsf{v}[\![\psi_1 \vee \psi_2]\!]$
1	1	1	1	1
1	0	0	1	1
0	1	1	0	1
0	0	1	1	0

Notice that there is only one row where columns 3, 4, and 5 are all 1; this corresponds the valuations where ψ_1 and ψ_2 evalaute to 1, and under every such valuation, all formulas in Γ are satisfied. In this row, since ψ_1 also evaluates to 1, we have that $\Gamma \models \psi_1$.

It is worth observing one special case of Definition 16 — when $\Gamma = \emptyset$. In this case, every valuation satisfies every formula in Γ (vaccuously, since there are none to satisfy). Therefore, if $\emptyset \models \varphi$, then every truth assignment satisfies φ. Such formulas are called *tautologies*, and they represent universal truths that hold in every model/world/assignment.

Definition 18 (Tautologies)**.** A wff φ is a *tautology* or is *valid* if for every valuation v, $\mathsf{v} \models \varphi$. In other words, $\emptyset \models \varphi$. We will denote this simply as $\models \varphi$.

Example 19. We will show $\varphi = \psi_1 \to (\psi_2 \to \psi_1)$ is a tautology, no matter what formulas ψ_1 and ψ_2 are. The proof is once again organized as truth table, and we show that in all rows the formula φ evaluates to 1.

$\mathsf{v}[\![\psi_1]\!]$	$\mathsf{v}[\![\psi_2]\!]$	$\mathsf{v}[\![\psi_2 \to \psi_1]\!]$	$\mathsf{v}[\![\psi_1 \to (\psi_2 \to \psi_1)]\!]$
1	1	1	1
1	0	1	1
0	1	0	1
0	0	1	1

The last important notion we would like to introduce is that of *satisfiability*.

Definition 20 (Satisfiable)**.** A formula φ is satisfiable if there is some valuation v such that $\mathsf{v} \models \varphi$. In other words, $[\![\varphi]\!] \neq \emptyset$.

Example 21. $\varphi = (p \to q) \to r$ is satisfiable because: Consider the valuation v that assigns 1 to every proposition. Now, since $\mathsf{v} \models r$, we have $\mathsf{v} \models \varphi$, based on the semantics of $\to$.

We conclude our introduction to propositional logic by consider two fundamental computational problems — *satisfiability* and *validity*.

Satifiability Given a formula φ, determine if φ is satisfiable.

Validity Given a formula φ, determine if φ is a tautology.

The satisfiability and validity problems have very simple algorithms to solve them. To check if φ, over propositions $\{p_1, \ldots p_n\}$, is satisfiable (is a tautology), compute $\mathsf{v}[\![A]\!]$ for every truth assignment v to the propositions $\{p_1, \ldots p_n\}$. The running time for this algorithm is $O(2^n)$. One of the most important open questions in computer science is whether this is the best algorithm for these problems. We will explore these problems in greater depth in the coming weeks.