CS 228: Logic in Computer Science

Amit Rajaraman

Last updated January 12, 2021

Contents

§1. Introduction to Logic

1.1. Lecture 1

For any computer scientist, logic is an extremely basic tool. Consider the statement

This sentence is false.

A little bit of thought shows that the above sentence has no definite truth value – it is *paradoxical*. Indeed, it is often known as the "liar's paradox". This sort of self-referential sentence will come back to haunt us many more times in the future.

Propositional logic (or *zeroth-order logic*) is basically the form of logic that deals with propositions which can be true or false as well as relations between them.
A more useful tool is that of *first-order logic*, that also deals with non-logical objects, predicates about them, and quantifiers ($\forall$ and $\exists$). That is, we are allowed to quantify over elements of the set, but not something like subsets of the set. A lot of mathematical statements cannot be written when we are restricted to first-order logic.

We study theories with basic assumptions or *axioms*. Using these axioms, we aim to prove more non-trivial results within the theory. A natural question to ask is: is it possible to have some set of axioms that allow us to concretely determine the truth value of any consequential statement? Once more, the self-referential statement returns.

Theorem 1.1 (Gödel's Incompleteness Theorem)**.** There are theories whose assumptions cannot be listed.

Proof. Suppose that there exists such a list for the number theory. Consider the **true** statement

This sentence cannot be proven by the list.

The list cannot imply the sentence. This yields a contradiction. ■

This theorem shows that logic has all but failed as a tool to do math. From this failure, rose computer science. This is the basic spirit of the proof and might not be completely understood. We discuss it more concretely later.

1.2. Lecture 2

1.2.1. Propositional Logic: Syntax and Parsing

We need an efficient method to identify if some group of symbols is a logical argument. We usually define a syntax for this (for example, grammar in English).

The logic we consider is over some list of propositions. We give each proposition a symbol. So say there is some set Vars of *countably many* propositional variables. These propositional variables are also called *Boolean variables.* Propositions are connected by *logical arguments.* How could we connect propositions?

- A statement that is always true/false.
- Negation. A statement that is the negation of another.
- Conjunction. Two statements being true simultaneously.
- Disjunction. At least one of two statements being true.
- Implication. If a statement is true, then some other statement is true as well.
- Equivalence. Two statements always have the same truth value.
- Disequality or exclusive or. Two statements always have different truth values.

true	$\top$	top
false	$\bot$	bot
negation	$\neg$	not
conjunction	$\wedge$	and
disjunction	$\vee$	or
implication	$\implies$	implies
equivalence	$\iff$	iff
exclusive or	$\oplus$	xor
opening parenthesis	(	
closing parenthesis	)	

We assume that the above *logical connectives* are not in Vars.
A *propositional formula* is a finite string containing symbols in Vars and logical connectives.

Definition 1.1. The set of propositional formulas is the smallest set P such that

- $\top, \bot \in P$,
- $\mathsf{Vars} \subseteq P$,
- if $f \in P$, then $\neg f \in P$, and
- if $\circ$ is a binary symbol and $f, g \in P$, then $(f \circ g) \in P$.

Alternatively, this can succinctly be written as "$f \in P$ if

$$f \coloneqq p \mid \top \mid \bot \mid \neg f \mid (f \vee f) \mid (f \wedge f) \mid (f \implies f) \mid (f \iff f) \mid (f \oplus f)$$

where $p \in \mathsf{Vars}$."

Definition 1.2. $\top$, $\bot$, and any $p \in \mathsf{Vars}$ are known as *atomic formulas.*

Definition 1.3. For each $f \in P$, $\mathsf{Vars}(f)$ is the set of variables appearing in f.

It is important to note that parentheses are needed (only) between binary operations. So as of now, $(\bot \implies \top)$ is a formula but $\bot \implies \top$ isn't.
Not all strings over Vars and logical connectives are in P.

1.2.2. Examples Encoding Arguments into Logic

Consider the following argument.

If c then if s then f. not f. Therefore, if s then not c.

This can be written as

$$(((c \implies (s \implies f)) \wedge \neg f) \implies (s \implies \neg c)).$$

Another example, say we know that good people always tell the truth and not good people always tell a lie. If there are two people A and B and A says "I am not good or B is good", then what are A and B?
Suppose the variables p_A and p_B denote whether A and B are truthful or not. Then the above is basically

$$((\neg p_A \vee p_B) \iff p_A).$$

How do we determine whether there is some p_A, p_B that satisfies this?

1.2.3. Parsing Formulas

$F \in P$ iff it can be obtained by unfolding one of these generation rules.

Definition 1.4. A *parse tree* of a formula $F \in P$ is a tree such that

- the root is F,
- the leaves are atomic formulas, and
- each internal node is formed by applying some formulation rule on its children.

We have the following

Theorem 1.2. *$F \in P$ iff there is a parse tree of F.*

Proof. The reverse direction follows by definition.
How do we show that any $F \in P$ has a parse tree? In fact, it has a *unique* parse tree. ■

A parse tree is a directed acyclic graph (DAG). The parsing produces a parse DAG. This is done by not writing repeated symbols twice, ensuring that all arrows go from higher levels of the DAG to the lower ones.

Definition 1.5. A formula G is a *subformula* of a formula F if G occurs within F. Further, G is a proper subformula of F if $F \neq G$. Denote by $\mathsf{sub}(F)$ the set of subformulas of F.

Observe that the nodes of the parse tree of F form $\mathsf{sub}(F)$.
Immediate subformulas are the children of a formula in its parse tree. The corresponding *leading connective* is the connective used to join the children. So for example,

$$\mathsf{sub}((\neg p_2 \iff (p_1 \wedge p_3)) = \{((\neg p_2 \iff (p_1 \wedge p_3)), \neg p_2, (p_1 \wedge p_3), p_1, p_2, p_3\}.$$

1.2.4. Shorthands

The reader might have noticed by now that we need to write so many parentheses, which don't really feel necessary most of the time. If we use some sort of precedence order over logical connectives, we may be able to drop some parentheses without losing the unique parsing property.
For example, we may drop outermost parentheses without any confusion. An example of this is writing $((p \wedge q) \implies (r \vee p))$ as $(p \wedge q) \implies (r \vee p)$.
Further, in the above example, if we give $\vee$ and $\wedge$ higher precedence then $\implies$ during parentheses, then we can drop all the parentheses! The usual precedence order we use is

$$\neg > \vee = \wedge = \oplus > \implies = \iff .$$

So how do we go about parsing a formula then? Suppose we have $F_0 \circ_1 F_2 \circ_2 \cdots \circ_n F_n$, where each F_i is either atomic, enclosed by parentheses, or their negation. We transform it as follows.

- Find a $\circ_i$ such that $\circ_{i-1}$ and $\circ_{i+1}$ have lower precedence (if they exist).
- Introduce parentheses around $F_{i-1} \circ F_i$ and call it $F_i' := (F_{i-1} \circ_i F_i)$ so you now have

$$F_0 \circ_1 \cdots \circ_{i-2} F_{i-2} \circ_{i-1} F_i' \circ_{i+1} F_{i+1} \circ_{i+2} \cdots \circ_n F_n.$$

Repeat the above until only one expression remains (n becomes 1). We can then parse it normally. For example,

$$p \wedge q \implies r \vee p \text{ to } (p \wedge q) \implies r \vee p \text{ to } (p \wedge q) \implies (r \vee p) \text{ to } ((p \wedge q) \implies (r \vee p)).$$

Some formulas cannot be unambiguously parsed, for example $p \vee q \wedge r$, $p \vee q \vee r$, or $p \implies q \implies r$. But can we salvage any of them?
Associativity preference may further reduce the need of parentheses. Let's make all our operators right associative (first group the rightmost occurrence). So for example, unless mentioned otherwise, we take $p \implies q \implies r$ as $(p \implies (q \implies r))$.

Definition 1.6. For $F \in P$ and $p_1, \ldots, p_k \in \mathsf{Vars}$, we denote by $F[G_1/p_1, \ldots, G_k/p_k]$ the formula obtained by *simultaneously* replacing all occurrences of p_i by the formula G_i for each $i \in [k]$.

So for example,

$$(p \implies (r \implies p))[(r \otimes p)/p] = ((r \otimes p) \implies (r \implies (r \otimes p))).$$

Sometimes, we may also write a formula F as $F(p_1, \ldots, p_k)$. Then, by $F(G_1, \ldots, G_n)$, we mean $F[G_1/p_1, \ldots, G_k/p_k]$.

1.3. Lecture 3

1.3.1. Semantics

Semantics is giving meaning to formulas. We denote the set of truth values as $\mathcal{B} \coloneqq \{0, 1\}$. We may view 0 as "false" and 1 as "true", but the only important thing is that they are distinct.

Definition 1.7 (Model)**.** A *model* is an function from $\mathsf{Vars} \to \mathcal{B}$.

For example, $\{p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, \ldots\}$. Since Vars is countable, the set of models is non-empty and infinite.
A given model m may or may not satisfy a formula F. This satisfaction relation is denoted by $m \vDash F$. Let us define this more concretely.

Definition 1.8. The *satisfaction relation* $\vDash$ between models and formulas is the smallest relation that satisfies the following.

- $m \vDash \top$,
- if $m(p) = 1$, then $m \vDash p$,
- if $m \nvDash F$, then $m \vDash \neg F$,
- if $m \vDash F_1$ or $m \vDash F_2$, then $m \vDash F_1 \vee F_2$,
- if $m \vDash F_1$ and $m \vDash F_2$, then $m \vDash F_1 \wedge F_2$,
- if $m \vDash F_1$ and $m \vDash F_2$ but not both, then $m \vDash F_1 \oplus F_2$,
- if if $m \vDash F_1$ then $m \vDash F_2$, then $m \vDash (F_1 \implies F_2)$, and
- if $m \vDash F_1$ iff $m \vDash F_2$, then $m \vDash (F_1 \iff F_2)$.

Observe that $\perp$ is not explicitly mentioned in the above definition since it follows from it being the *smallest* relation.

If $m \vDash F$, we say that m *satisfies* F.
F is *satisfiable* if there is a model m such that $m \vDash F$. This is often abbreviated as *sat.*
F is *valid* (written $\vDash F$) if for each model m, $m \vDash F$. A valid formula is also called a *tautology.*
F is *unsatisfiable* (written $\nvDash F$) if there is no model m such that $m \vDash F$. This is often abbreviated as *unsat.*

We can check if a certain formula satisfies a model by moving bottom-up in the parse tree.

We overload the $\vDash$ operator in several natural ways.

Definition 1.9. Let M be a set of models. We write $M \vDash F$ if for every $m \in M$, $m \vDash F$.

Definition 1.10. Let Σ be a set of formulas. We write $\Sigma \vDash F$ if for every m that satisfies every formula in Σ, $m \vDash F$.

This is read "Σ implies F". If $\Sigma = \{G\}$, we write $G \vDash F$.

Definition 1.11. We write $F \equiv G$ if for each model m,

$$m \vDash F \iff m \vDash G.$$

Definition 1.12. Formulas F and G are *equisatisfiable* if

$$F \text{ is sat} \iff G \text{ is sat.}$$

Definition 1.13. Formulas F and G are *equivalid* if $\vDash F \iff \vDash G$.

1.3.2. Decidability of SAT

Definition 1.14. A problem is *decidable* if there is an algorithm to solve the problem.

This is required since Gödel's Incompleteness implies the existence of undecidable problems.
The problem we consider here, known as the *propositional satisfiability problem* is:

For a given $F \in P$, is F satisfiable?

Theorem 1.3. The propositional satisfiability problem is decidable.

Proof. We enumerate the $2^{|\mathsf{Vars}|}$ elements of $\mathsf{Vars}(F) \to \mathcal{B}$. If any of the models satisfy the formula, F is sat. Otherwise, it is unsat. ■

The cost is obviously exponential and we would want to do better. There are, however, several tricks that make satisfiability checking more feasible for real-world formulas.

1.3.3. Truth Tables

We wish to assign a truth value to every formula F.
Given a model $m : \mathsf{Vars} \to \mathcal{M}$, we can naturally extend it to $m : P \to B$ as

$$m(F) = \begin{cases} 1, & m \vDash F, \\ 0, & \text{otherwise.} \end{cases}$$

This extended m is known as the *truth function*. We needn't introduce new symbols since this is a very natural extension.

For a formula F, a truth table consists f $2^{|\mathsf{Vars}(F)|}$ rows, where each row considers one of the models and computes the corresponding truth value of F.
For example, show that $p \vee q \equiv \neg(\neg p \wedge \neg q)$ and $p \wedge q \equiv \neg(\neg p \vee \neg q)$, also known as De Morgan's laws.
It is also easily shown that $p \implies q \equiv (\neg p \vee q)$.
Truth tables are tedious because we need to write 2^n rows even if a simple observation could easily show (un)satisfiability.
For example, $a \vee (c \neg a)$ is very clearly true. If there's no $\neg$s (of any form, $\oplus$ in particular) in general, one can just set everything as true. Another example is that $(a \vee (c \neg a)) \wedge \neg(a \vee (c \neg a))$ is obviously unsat.
How do we take such shortcuts?

1.3.4. Expressive power of propositional logic

A finite boolean function is one from $\mathcal{B}^n \to \mathcal{B}$.
A formula F with $\mathsf{Vars}(F) = \{p_1, \ldots, p_n\}$ can be viewed as a boolean function f such that for each model m, $m(F) = f(m(p_1), \ldots, m(p_n))$. This is just an alternate way of writing a truth table (as a function instead of a table).

Theorem 1.4. FOr each finite boolean function f, there is a formula F that represents f.

Proof. Let $f : \mathcal{B}^n \to \mathcal{B}$. Let $p_i^0 := \neg p_i$ and $p_i^1 := p_i$. For every $(b_1, \ldots, b_n) \in \mathcal{B}^n$, let

$$F_{(b_1,\ldots,b_n)} := \begin{cases} (p_1^{b_1} \wedge \cdots \wedge p_n^{b_n}), & f(b_1, \ldots, b_n) = 1 \\ \bot, & \text{otherwise.} \end{cases}$$

We can then define the required formula F by taking the conjunction over all boolean combinations,

$$F := F_{(0,\ldots,0)} \vee \cdots \vee F_{(1,\ldots,1)}.$$

■

Observe that we have only used three logical connectives.
What if we do not have all logical connectives? Then we may not be able to represent all boolean functions. This is known as "insufficient expressive power".

For example, $\wedge$ alone cannot express all boolean functions. Consider the function $f = \{0 \mapsto 1, 1 \mapsto 1\}$. We show that this cannot be achieved by any $\wedge$s by taking induction on the size of formulas containing the variable p and $\wedge$. For the base case, our only choice of formula is p. Now, suppose that formulas F and G of size less than $n-1$ do not represent f. We can construct a longer formula by $(F \wedge G)$. This formula does not represent f because we can always pick a model where F and G produce 0.
We originally used 8 connectives. This is not the minimal set required for maximum expressivity, however. For example, $\neg$ and $\vee$ can define the whole propositional logic. Indeed,

- $\top \equiv p \vee \neg p$,
- $\bot \equiv \neg\top$,
- $(p \wedge q) \equiv \neg(p \vee q)$,
- $(p \otimes q) \equiv (p \wedge \neg q) \vee (\neg p \wedge q)$,
- $(p \implies q) \equiv (\neg p \vee q)$, and
- $(p \iff q) \equiv (p \implies q) \wedge (q \implies p)$.