CS 228: Logic in Computer Science

Amit Rajaraman

Last updated February 20, 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.

The proof roughly goes by considering the self-referential true sentence

This sentence cannot be proven by the list.

and arriving at a contradiction.

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 (similar to 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 can 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

- $\top, \bot$ are 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

$$F \coloneqq p \mid \top \mid \bot \mid \neg F \mid (F \vee F) \mid (F \wedge F) \mid (F \Rightarrow F) \mid (F \Leftrightarrow F) \mid (F \oplus F)$$

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, $(\bot \Rightarrow \top)$ is a formula but $\bot \Rightarrow \top$ 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

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

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 are 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. Further, if $F \in P$, it has a *unique* parse tree.

The reverse direction follows by definition.

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 that joins it to 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 far more parentheses than required 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 $((p \wedge q) \Rightarrow (r \vee p))$ as $(p \wedge q) \Rightarrow (r \vee p)$.
Further, in the above example, if we give $\vee$ and $\wedge$ higher precedence then $\Rightarrow$ during parentheses, then we can drop all the parentheses! The usual precedence order we use is

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

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 $F_i' \coloneqq (F_{i-1} \circ_i F_i)$ so we 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 term remains. We can then parse it normally. For example,

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

Some formulas cannot be unambiguously parsed, for example $p \vee q \wedge r$, $p \vee q \vee r$, or $p \Rightarrow q \Rightarrow r$. 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 $p \Rightarrow q \Rightarrow r$ as $(p \Rightarrow (q \Rightarrow r))$.

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

So for example,

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

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

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, $\{p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, \dots\}$ is a model.
It is quite natural to extend this further to formulas in general. That is, a given model m may or may not satisfy a formula F. More concretely.

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

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

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

If $m \models F$, we say that m *satisfies* F.
F is *satisfiable* if there is a model m such that $m \models F$. This is often abbreviated as *sat*.
F is *valid* (written $\models F$) if for each model m, $m \models F$. A valid formula is also called a *tautology*.
F is *unsatisfiable* (written $\not\models F$) if there is no model m such that $m \models F$. This is often abbreviated as *unsat*.

A few things to note are:

- A formula is valid iff its negation is unsat.

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

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

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

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

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

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

$$m \models F \iff m \models G.$$

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 $\models F \iff \models G$.

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 $2^{|\mathsf{Vars}(F)|}$ 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. Indeed, there are 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 $m : \mathsf{Vars} \to \mathcal{B}$, we can naturally extend it to $m : P \to \mathcal{B}$ as

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

This extended m is known as the *truth function*. We do not introduce new notation for this, keeping the meanings from the definitions for models unchanged.

For a formula F, a truth table consists of $2^{|\mathsf{Vars}(F)|}$ rows, where each row considers one of the models and computes the corresponding truth value of F.
Truth tables are sometimes useful to prove that formulae are equivalent. 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 $p \Rightarrow q \equiv (\neg p \vee q)$.
Truth tables are tedious because we need to write 2^n rows even if a simple observation could easily show (un)satisfiability.
For example, $a \vee (c \wedge b)$ being sat is very clearly true. If there are 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 $p_i^0 \coloneqq \neg p_i$ and $p_i^1 \coloneqq p_i$. For every $(b_1, \ldots, b_n) \in \mathcal{B}^n$, let

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

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

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

∎

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 or 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,

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

1.4. Lecture 4

1.4.1. Formal Proofs

Suppose that for a set of formulas Σ and a formula F, $\Sigma \models F$. Can we infer that $\Sigma \models F$ without writing out the truth tables? This syntactic inference is called *derivation*. This is written $\Sigma \vdash F$ and is read "Σ proves F". In this case, F is said to be a "consequence" of Σ.

If F occurs on the left hand side $F \in \Sigma$, then F is clearly a consequence.

A *proof rule* provides us a means to derive new statements from old statements. They are written as

$$\mathsf{RuleName}\ \frac{\text{Stuff already there}}{\text{Stuff to be added}}\ \mathsf{Conditions\ to\ be\ met}$$

A derivation proceeds by applying these proof rules. So for instance, the rule we mentioned earlier can be written as

$$\mathsf{Assumption}\ \frac{}{\Sigma \vdash F}\ F \in \Sigma.$$

Another obvious example is

$$\mathsf{Monotonic}\ \frac{\Sigma \vdash F}{\Sigma' \vdash F}\ \Sigma \subseteq \Sigma'.$$

Definition 1.15. A *derivation* is a list of statements that are derived from earlier statements.

An example of a derivation using the above rules is:

1. $\{p \vee q, \neg\neg q\} \vdash \neg\neg q$ (Assumption)
2. $\{p \vee q, \neg\neg q, r\} \vdash \neg\neg q$ (Monotonic applied to 1)

It is important to note that we need to explicitly point out which earlier step we are using (if any).
Let us try to establish some proof rules on our logical connectives.

Negation.

$$\textsf{DoubleNeg}\ \frac{\Sigma \vdash F}{\Sigma \vdash \neg\neg F}$$

For example,

1. $\{p \vee q, r\} \vdash r$ (Assumption)
2. $\{p \vee q, r, \neg\neg q\} \vdash r$ (Monotonic applied to 1)
3. $\{p \vee q, r, \neg\neg q\} \vdash \neg\neg r$ (DoubleNeg applied to 2)

Conjunction. We have the following proof rules for the conjunction.

$$\wedge\textsf{-Intro}\ \frac{\Sigma \vdash F \qquad \Sigma \vdash G}{\Sigma \vdash F \wedge G}$$

$$\wedge\textsf{-Elim}\ \frac{\Sigma \vdash F \wedge G}{\Sigma \vdash F}$$

$$\wedge\textsf{-Symm}\ \frac{\Sigma \vdash F \wedge G}{\Sigma \vdash G \wedge F}$$

An example using these is

1. $\{p \wedge q, \neg\neg q, r\} \vdash p \wedge q$ (Assumption)
2. $\{p \wedge q, \neg\neg q, r\} \vdash p$ ($\wedge$-Elim applied to 1)
3. $\{p \wedge q, \neg\neg q, r\} \vdash q \wedge p$ ($\wedge$-Symm applied to 1)

Disjunction. Except for the last two (which are like De Morgan's law), the rules for the disjunction are similar.

$$\vee\textsf{-Intro}\ \frac{\Sigma \vdash F}{\Sigma \vdash F \vee G}$$

$$\vee\textsf{-Elim}\ \frac{\Sigma \vdash F \vee G \qquad \Sigma \cup \{F\} \vdash H \qquad \Sigma \cup \{G\} \vdash H}{\Sigma \vdash H}$$

$$\vee\textsf{-Symm}\ \frac{\Sigma \vdash F \vee G}{\Sigma \vdash G \vee F}$$

$$\vee\textsf{-Def}\ \frac{\Sigma \vdash F \vee G}{\Sigma \vdash \neg(\neg F \wedge \neg G)}$$

$$\vee\textsf{-Def}\ \frac{\Sigma \vdash \neg(\neg F \wedge \neg G)}{\Sigma \vdash F \vee G}$$

Let us give another example, which is basically equivalent to the distributive law. Suppose we have $\Sigma \vdash (F \wedge G) \vee (F \wedge H)$, we want to show that we can derive $\Sigma \vdash F \wedge (G \vee H)$. Indeed,

1. $\Sigma \vdash (F \wedge G) \vee (F \wedge H)$ (Premise)
2. $\Sigma \cup \{F \wedge G\} \vdash F \wedge G$ (Assumption)
3. $\Sigma \cup \{F \wedge G\} \vdash F$ ($\wedge$-Elim applied to 2)
4. $\Sigma \cup \{F \wedge G\} \vdash G \wedge F$ ($\wedge$-Symm applied to 2)
5. $\Sigma \cup \{F \wedge G\} \vdash G$ ($\wedge$-Elim applied to 4)
6. $\Sigma \cup \{F \wedge G\} \vdash G \vee H$ ($\vee$-Intro applied to 5)
7. $\Sigma \cup \{F \wedge G\} \vdash F \wedge (G \vee H)$ ($\wedge$-Intro applied to 3,6)
8. $\Sigma \cup \{F \wedge H\} \vdash F \wedge H$ (Assumption)
9. $\Sigma \cup \{F \wedge H\} \vdash F$ ($\wedge$-Elim applied to 8)
10. $\Sigma \cup \{F \wedge H\} \vdash H \wedge F$ ($\wedge$-Symm applied to 8)
11. $\Sigma \cup \{F \wedge H\} \vdash H$ ($\wedge$-Elim applied to 10)
12. $\Sigma \cup \{F \wedge H\} \vdash H \vee G$ ($\vee$-Intro applied to 11)
13. $\Sigma \cup \{F \wedge H\} \vdash G \vee H$ ($\vee$-Symm applied to 11)
14. $\Sigma \cup \{F \wedge H\} \vdash F \wedge (G \vee H)$ ($\wedge$-Intro applied to 9,13)
15. $\Sigma \vdash F \wedge (G \vee H)$ ($\vee$-Elim applied to 1,7,14)

Implication. The rules for the implication are:

$$\Rightarrow\text{-Intro}\ \frac{\Sigma \cup \{F\} \vdash G}{\Sigma \vdash F \Rightarrow G}$$

$$\Rightarrow\text{-Elim}\ \frac{\Sigma \vdash F \Rightarrow G \qquad \Sigma \vdash F}{\Sigma \vdash G}$$

$$\Rightarrow\text{-Def}\ \frac{\Sigma \vdash F \Rightarrow G}{\Sigma \vdash \neg F \vee G}$$

$$\Rightarrow\text{-Def}\ \frac{\Sigma \vdash \neg F \vee G}{\Sigma \vdash F \Rightarrow G}$$

For example, let us show that $\{\neg p \vee q, p\} \vdash q$.

1. $\{\neg p \vee q, p\} \vdash p$ (Assumption)
2. $\{\neg p \vee q, p\} \vdash \neg p \vee q$ (Assumption)
3. $\{\neg p \vee q, p\} \vdash p \Rightarrow q$ ($\Rightarrow$-Def applied to 2)
4. $\{\neg p \vee q, p\} \vdash q$ ($\Rightarrow$-Elim applied to 1,3)

As another example, let us show that $\varnothing \vdash (p \Rightarrow q) \vee p$.

1. $\{\neg p\} \vdash \neg p$ (Assumption)
2. $\{\neg p\} \vdash \neg p \vee q$ ($\vee$-Intro applied to 1)
3. $\{\neg p\} \vdash p \Rightarrow q$ ($\Rightarrow$-Def applied to 2)

4. $\{\neg p\} \vdash (p \Rightarrow q) \vee p$ ($\vee$-Intro applied to 3)
5. $\{p\} \vdash p$ (Assumption)
6. $\{p\} \vdash p \vee (p \Rightarrow q)$ ($\vee$-Intro applied to 5)
7. $\{p\} \vdash (p \Rightarrow q) \vee p$ ($\vee$-Symm applied to 6)
8. $\varnothing \vdash (p \Rightarrow p)$ ($\Rightarrow$-Intro applied to 5)
9. $\varnothing \vdash (\neg p \vee p)$ ($\Rightarrow$-Def applied to 8)
10. $\varnothing \vdash (p \Rightarrow q) \vee p$ ($\vee$-Elim applied to 4,7,9)

There are several more proof rules for parentheses, $\oplus$, $\iff$, et cetera.

1.4.2. Soundness

How do we know that the above proof rules are correct? What does "correct" even mean?

Theorem 1.5. If proof rules derive a statement $\Sigma \vdash F$, then $\Sigma \models F$.

We use an inductive argument. Assume that the theorem holds for the premises of the rules. We shall show that it is also true for the conclusions.

The above basically unifies the *syntactic* and *semantic* methods of proof. We shall later see that the converse is true holds as well.
Consider the following rule

$$\wedge\text{-Elim}\ \frac{\Sigma \vdash F \wedge G}{\Sigma \vdash F}$$

Consider a model $m \models \Sigma$. By the induction hypothesis, $m \models F \wedge G$. It is easy to show (using the truth table) that if $m \models F \wedge G$, then $m \models F$. Therefore, $\Sigma \models F$.

As another example, consider

$$\Rightarrow\text{-Intro}\ \frac{\Sigma \cup \{F\} \vdash G}{\Sigma \vdash F \Rightarrow G}$$

Consider a model $m \models \Sigma$. There are two possibilities.

- $m \models F$. Then $m \models \Sigma \cup \{F\}$. By the hypothesis, $m \models G$. Therefore, $m \models F \Rightarrow G$.
- $m \not\models F$. Then m trivially satisfies $m \models F \Rightarrow G$.

1.5. Lecture 5

1.5.1. Derived Rules

In this section and the next, we give some rules derived from those already given that are quite useful when trying to prove statements.

Modus ponens.

$$\vee\text{-ModusPonens}\ \frac{\Sigma \vdash \neg F \vee G \qquad \Sigma \vdash F}{\Sigma \vdash G}$$

The proof is as follows.

1. $\Sigma \vdash \neg F \vee G$ (Premise)
2. $\Sigma \vdash (F \Rightarrow G)$ ($\Rightarrow$-Def applied to 1)
3. $\Sigma \vdash F$ (Premise)
4. $\Sigma \vdash G$ ($\Rightarrow$-Elim applied to 2,3)

Tautology.

$$\text{Tautology}\ \frac{}{\Sigma \vdash \neg F \vee F}$$

This can be derived as

1. $\Sigma \cup \{F\} \vdash F$ (Assumption)
2. $\Sigma \vdash (F \Rightarrow G)$ ($\Rightarrow$-Intro applied to 1)
3. $\Sigma \vdash \neg F \vee F$ ($\Rightarrow$-Def applied to 2)

Contradiction.

$$\text{Contra}\ \frac{\Sigma \vdash F \wedge \neg F}{\Sigma \vdash G}$$

This is proved as

1. $\Sigma \vdash (F \wedge \neg F)$ (Premise)
2. $\Sigma \vdash (\neg F \wedge G)$ ($\wedge$-Symm applied to 1)
3. $\Sigma \vdash \neg F$ ($\wedge$-Elim applied to 2)
4. $\Sigma \vdash \neg F \vee G$ ($\vee$-Intro applied to 3)
5. $\Sigma \vdash F$ ($\wedge$-Elim applied to 1)
6. $\Sigma \vdash G$ ($\vee$-ModusPonens applied to 4,5)

Contrapositive.

$$\text{Contrapositive}\ \frac{\Sigma \cup \{F\} \vdash G}{\Sigma \cup \{\neg G\} \vdash \neg F}$$

This is proved as

1. $\Sigma \cup \{F\} \vdash G$ (Premise)
2. $\Sigma \cup \{F\} \vdash \neg\neg G$ (DoubleNeg applied to 1)
3. $\Sigma \vdash (F \Rightarrow \neg\neg G)$ ($\Rightarrow$-Intro applied to 2)
4. $\Sigma \vdash \neg F \vee \neg\neg G$ ($\Rightarrow$-Def applied to 3)
5. $\Sigma \vdash \neg\neg G \vee \neg F$ ($\vee$-Symm applied to 4)
6. $\Sigma \vdash \neg G \Rightarrow \neg F$ ($\Rightarrow$-Def applied to 5)
7. $\Sigma \cup \{\neg G\} \vdash \neg G \Rightarrow \neg F$ (Monotonic applied to 6)
8. $\Sigma \cup \{\neg G\} \vdash \neg G$ (Assumption)
9. $\Sigma \cup \{\neg G\} \vdash \neg F$ ($\Rightarrow$-Elim 7,8)

1.5.2. More Derived Rules

Proof by cases.

$$\text{ByCases}\ \frac{\Sigma \cup \{F\} \vdash G \qquad \Sigma \cup \{\neg F\} \vdash G}{\Sigma \vdash G}$$

This can be proved as

1. $\Sigma \cup \{F\} \vdash G$ (Premise)
2. $\Sigma \cup \{\neg F\} \vdash G$ (Premise)
3. $\Sigma \vdash F \vee \neg F$ (Tautology)
4. $\Sigma \vdash G$ ($\vee$-Elim applied to 1,2,3)

Proof by contradiction.

$$\text{ByContra}\ \frac{\Sigma \cup \{F\} \vdash G \qquad \Sigma \cup \{F\} \vdash \neg G}{\Sigma \vdash \neg F}$$

This is proved as

1. $\Sigma \cup \{F\} \vdash G$ (Premise)
2. $\Sigma \cup \{F\} \vdash \neg G$ (Premise)
3. $\Sigma \cup \{\neg G\} \vdash \neg F$ (Contrapositive applied to 1)
4. $\Sigma \cup \{\neg\neg G\} \vdash \neg F$ (Contrapositive applied to 1)
5. $\Sigma \vdash \neg F$ (ByCases applied to 3,4)

Reverse Double Negation.

$$\text{RevDoubleNeg}\ \frac{\Sigma \vdash \neg\neg F}{\Sigma \vdash F}$$

This can be proved as

1. $\Sigma \vdash \neg\neg F$ (Premise)
2. $\Sigma \cup \{\neg F\} \vdash \neg\neg F$ (Monotonic applied to 1)
3. $\Sigma \cup \{\neg F\} \vdash \neg F$ (Assumption)
4. $\Sigma \cup \{\neg F\} \vdash \neg F \wedge \neg\neg F$ ($\wedge$-Intro applied to 2,3)
5. $\Sigma \cup \{\neg F\} \vdash \neg F$ ($\wedge$-Elim applied to 4)
6. $\Sigma \cup \{\neg F\} \vdash \neg\neg F \wedge \neg F$ ($\wedge$-Symm applied to 5)
7. $\Sigma \cup \{\neg F\} \vdash F$ (Contra applied to 6)
8. $\Sigma \cup \{F\} \vdash F$ (Assumption)
9. $\Sigma \vdash F$ (ByCases applied to 7,8)

Resolution.

$$\text{Resolution}\ \frac{\Sigma \vdash \neg F \vee G \qquad \Sigma \vdash F \vee H}{\Sigma \vdash G \vee H}$$

This is proved as

1. $\Sigma \vdash \neg F \vee G$ (Premise)
2. $\Sigma \cup \{F\} \vdash \neg F \vee G$ (Monotonic applied to 1)
3. $\Sigma \cup \{F\} \vdash F$ (Assumption)
4. $\Sigma \cup \{F\} \vdash G$ ($\vee$-ModusPonens applied to 2,3)
5. $\Sigma \cup \{F\} \vdash G \vee H$ ($\vee$-Intro applied to 4)
6. $\Sigma \vdash F \vee H$ (Premise)
7. $\Sigma \cup \{\neg F\} \vdash F \vee H$ (Monotonic applied to 6)
8. $\Sigma \cup \{\neg F\} \vdash \neg F$ (Assumption)
9. $\Sigma \cup \{F\} \vdash H$ ($\vee$-ModusPonens applied to 7,8)
10. $\Sigma \cup \{\neg F\} \vdash H \vee G$ ($\vee$-Intro applied to 9)
11. $\Sigma \cup \{\neg F\} \vdash G \vee H$ ($\vee$-Symm applied to 10)
12. $\Sigma \vdash G \vee H$ (ByCases applied to 5,11)

1.5.3. Substitutions

Let $F_1(p)$ and $F_2(p)$ be formulas. If $\Sigma \vdash (F_1(G) \Leftrightarrow F_1(H))$, $\Sigma \vdash (F_2(G) \Leftrightarrow F_2(H))$, and $\Sigma \vdash F_1(G) \wedge F_2(G)$, then $\Sigma \vdash F_1(H) \wedge F_2(H)$.
The proof of the above is

1. $\Sigma \vdash (F_1(G) \Leftrightarrow F_1(H))$ (Premise)
2. $\Sigma \vdash (F_2(G) \Leftrightarrow F_2(H))$ (Premise)
3. $\Sigma \vdash (F_1(G) \wedge F_2(G))$ (Premise)
4. $\Sigma \vdash F_1(G)$ ($\wedge$-Elim applied to 3)
5. $\Sigma \vdash (F_1(G) \Rightarrow F_1(H))$ ($\Leftrightarrow$-Def applied to 1)
6. $\Sigma \vdash F_1(H)$ ($\Rightarrow$-Elim applied to 4,5)
7. $\Sigma \vdash F_2(G) \wedge F_1(G)$ ($\wedge$-Symm applied to 3)
8. $\Sigma \vdash F_2(G)$ ($\wedge$-Elim applied to 7)
9. $\Sigma \vdash (F_2(G) \Rightarrow F_2(H))$ ($\Leftrightarrow$-Def applied to 2)
10. $\Sigma \vdash F_2(H)$ ($\Rightarrow$-Elim applied to 8,9)
11. $\Sigma \vdash F_1(H) \wedge F_2(H)$ ($\wedge$-Intro applied to 6,10)

Let $F(p)$ be a formula. If $\Sigma \vdash (G \Leftrightarrow H)$ and $\Sigma \vdash F(G)$, then $\Sigma \vdash F(H)$.
This is easily shown by using arguments similar to the one above for each connective and then using induction.

However, we do not introduce substitution as a rule since it causes a lot of overhead for the proof-checker. We want the time cost of the proof-checker to be low, and allowing substitutions may not allow that to happen. So for example, an argument such as

1. $\Sigma \vdash \neg(\neg\neg F \vee G)$ (Premise)
2. $\Sigma \vdash \neg(F \vee G)$ (DoubleNeg applied to $\neg\neg F$ in 1)

is disallowed for our purposes.

So far, we have seen several rules of reasoning. But how would one determine if these rules are sufficient to derive all true statements? Further, this feels quite inconvenient since there are so many rules. Often, we also don't know where to start applying them to a particular problem. We aim to simplify and algorithmize this process.

1.6. Lecture 6

If we want to develop algorithms, we require more structure in our input. We need to figure out methods for simplification and suitable "normal forms".
In order to prove theorems, we need to get used to "structural induction". That is,

Theorem 1.6 (Structural Induction)**.** Every formula in P has property Q if

- every atomic formula has Q and
- if $F, G \in P$ have Q then so do $\neg F$ and $(F \circ G)$, where $\circ$ is any binary symbol.

Substitution is an important part of logic. We should be able to substitute equivalent subformulas without altering the truth values of formulas.

Lemma 1.7. Suppose we have formulas $F(p)$, G, and H. Suppose for some model m, $m \vDash G \Leftrightarrow m \vDash H$. Then $m \vDash F(G) \Leftrightarrow m \vDash F(H)$.

This can be shown using structural induction, working it out for each connective.
We shall use the above "substitution theorem" quite a lot in proofs.

Lemma 1.8. If $F(p) \equiv G(p)$, then for each formula H, $F(H) \equiv G(H)$.

We may assume without loss of generality that p does not appear in H (Why?). Consider a model m. Then define the model m' by

$$m' := \begin{cases} m[p \mapsto 1], & m \vDash H, \\ m[p \mapsto 0], & m \nvDash H. \end{cases}$$

Observe that by construction, $m' \vDash p$ iff $m' \vDash H$.
Further note that since p does not appear in $F(H)$, $m \vDash F(H)$ iff $m' \vDash F(H)$. By the earlier lemma, $m' \vDash F(H)$ iff $m' \vDash F(p)$. Since $F(p) \equiv G(p)$, $m' \vDash F(p)$ iff $m' \vDash G(p)$.
Again going backwards like we did now, we see that $m' \vDash G(p)$ iff $m' \vDash G(H)$ iff $m \vDash G(H)$. Therefore, $m \vDash F(H)$ iff $m \vDash G(H)$.

Theorem 1.9. Let G, H, and $F(p)$ be formulas. If $G \equiv H$, then $F(G) \equiv F(H)$.

This is just a reformulation of Lemma 1.7.
This allows us to use known equivalences to modify formulas. This was also proved in the previous lecture using the proof system.

1.7. Lecture 7

1.7.1. Negation Normal Form

Observe the following equivalences that remove $\oplus$, $\Rightarrow$, and $\Leftrightarrow$ from formulas.

$$(p \Rightarrow q) \equiv (\neg p \vee q)$$
$$(p \oplus q) \equiv (p \vee q) \wedge (\neg p \vee \neg q)$$
$$(p \Leftrightarrow q) \equiv \neg(p \oplus q)$$

Removing either $\oplus$ or $\Leftrightarrow$ results in an explosion in the formula size, but we assume that we can remove them on will. Motivated by this, we define

Definition 1.16. A formula is in *negation normal form* (NNF) if $\neg$ appears only in front of the propositional variables.

For any formula F, there is a formula F' in NNF such that $F \equiv F'$.
We can do stuff like $\neg(p \vee q) = (p \wedge \neg q)$ for every logical connective and push $\neg$ under them.

There are also $\neg$s hidden inside $\oplus$, $\Rightarrow$, and $\Leftrightarrow$. It is advisable to also remove these when producing the NNF of a formula.

1.7.2. Conjunctive Normal Form

Propositional variables are also referred to as *atoms* (recall "atomic formulas"). A *literal* is either an atom or its negation. A *clause* is a disjunction of literals.
Since $\vee$ is associative, commutative, and absorbs multiple occurrences, a clause can be referred to as a set of literals.

Definition 1.17. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses.

Since $\wedge$ is associative, commutative, and absorbs multiple occurrences, a CNF formula can be referred to as a set of clauses.
For any formula F, there is a formula F' in NNF such that $F \equiv F'$.
First, we remove $\oplus$, $\Rightarrow$, and $\Leftrightarrow$ using the usual equivalences. We then convert it to NNF. We then "flatten" $\vee$ and $\wedge$ using the associativity of $\vee$ and $\wedge$. The parse tree of the formula then has $\vee$ and $\wedge$ at alternating levels with the leaves being literals. We then distribute $\vee$ over $\wedge$ while flattening at each step to obtain a formula in CNF.
But does this complete the argument? No, we also need to show that the final part of the algorithm (distribution) terminates (pushing using distributivity increases the size of the formula). To see why it should terminate, define $\nu(F)$ to be the maximum height of the $\vee$-$\wedge$ alternations in F. Suppose there is a subformula G such that

$$G = \bigvee_{i=0}^{m} \bigwedge_{j=0}^{n_i} G_{ij}$$

After pushing, we obtain

$$G' = \bigwedge_{j_1=0}^{n_1} \cdots \bigwedge_{j_m=0}^{n_m} \bigvee_{i=0}^{m} G_{ij}.$$

Note that the height of the height of the above must be strictly less than $\nu(G)$.
Observe that G' is either the top formula or the parent of this connective is $\wedge$. Also, each G_{ij} is either a literal or a clause.
We must keep flattening to keep $F(G')$ in the considered form. Due to König's lemma, this procedure must terminate.

Lemma 1.10 (König's Lemma)**.** For an infinite connected graph G, if each node has finite degree, then there is an infinite simple path from each node.

A *unit clause* contains only one literal. A *binary clause* contains exactly two literals. A *ternary clause* contains exactly three literals.
We can also extend this to the empty set of literals. We take by convention that $\perp$ is the empty clause.

1.7.3. Tseitin's Encoding

CNF is desirable because it has a very single structure, and only involves three distinct connectives ($\vee$, $\wedge$, and $\neg$). The transformation using distributivity is bad because it leads to an explosion in the size of the formula. How do we avoid this?
By introducing fresh variables, Tseitin's encoding translates any formula into an equisatisfiable formula in CNF without an exponential explosion.

1. First, assume the input formula F is in NNF without $\oplus$, $\Rightarrow$, and $\Leftrightarrow$.
2. Find a $G_1 \wedge \cdots \wedge G_n$ in F that is just below an $\vee$. If no such subformula exists, terminate.
3. Replace $F(G_1 \wedge \cdots \wedge G_n)$ by $F(p) \wedge (\neg p \vee G_1) \wedge \cdots \wedge (\neg p \vee G_n)$, where p is a newly introduced variable.
4. Go to step 2.

For example, $(p_1 \wedge \cdots \wedge p_n) \vee (q_n \wedge \cdots \wedge q_m)$ becomes

$$(x \vee y) \wedge \bigwedge_{1 \leq i \leq n} (\neg x \vee p_i) \wedge \bigwedge_{1 \leq j \leq m} (\neg y \vee q_j),$$

which has only $m + n + 1$ clauses, as opposed to the mn clauses returned by the naïve algorithm. Here, x and y are the newly introduced variables.

Why does this preserve satisfiability?
Suppose $m \vDash F(p) \wedge (\neg p \vee G_1) \wedge \cdots \wedge (\neg p \vee G_n)$. We have three cases.

(i) If $m \vDash p$, then $m \vDash G_i$ for every $1 \leq i \leq n$. Therefore, $m \vDash G_1 \wedge \cdots \wedge G_n$. We can then perform a substitution to get that $m \vDash F(G_1 \wedge \cdots \wedge G_n)$.

(ii) If $m \nvDash p$ and $m \nvDash G_1 \wedge \cdots \wedge G_n$. Since $m \vDash F(p)$, we can directly apply the substitution theorem again.

(iii) If $m \nvDash p$ and $m \vDash G_1 \wedge \cdots \wedge G_n$. Since $F(G_1 \wedge \cdots \wedge G_n)$ is in NNF, p must occur positively (no $\neg p$) in $F(p)$. Therefore, $m[p \mapsto 1] \vDash F(p)$. Since p does not occur in any G_i, we also obviously have $m[p \mapsto 1] \vDash G_1 \wedge \cdots \wedge G_n$. Therefore, $m \vDash F(G_1 \wedge \cdots \wedge G_n)$ (we basically ignore the modification).

It is worth noting that going from a model that satisfies the encoding to one that satisfies the original formula is non-trivial (if it was, we might have $P = NP$ - Why?).

1.7.4. Disjunctive Normal Form

Similar to CNF, we define DNF as:

Definition 1.18. A formula is said to be in Disjunctive Normal Form (DNF) if it is a disjunction of conjunctions of literals.

1.8. Lecture 8

1.8.1. k-sat

Definition 1.19. A k-sat formula is a CNF formula with at most k literals in each clause.

For example, $(p \wedge q \wedge \neg r)$ is 1-sat.

Theorem 1.11. For any k-sat formula F, there is a 3-sat formula F' with linear blow up such that F and F' are equisatisfiable.

Proof. Let F be k-sat with $k \geq 4$. Let $G = (\ell_1 \vee \cdots \ell_k)$ be a clause, where each ℓ_i is a literal.
Let $x_1, \ldots, x_{k-2}$ be variables that don't appear in F and let G' be the set of clauses

$$(\ell_1 \vee \ell_2 \vee x_2) \wedge \bigwedge_{2 \leq i \leq k-3} (\neg x_i \vee x_{i+1} \vee \ell_{i+1}) \wedge (\neg x_{k-2} \vee \ell_{k-2} \vee \ell_k).$$

We claim that G and G' are equisatisfiable. Let $m \vDash G'$. Assume instead that $m(\ell_i) = 0$ for all i. We have $m(x_2) = 1$, and for each $2 \leq i \leq k-3$, $m(x_{i+1}) = 1$ (by induction). This implies that $m(x_{i+2}) = 1$, which results in a contradiction in the final clause. Therefore, $m \vDash G$.
For the other direction, let $m(\ell_i) = 1$. Then the model

$$m' = \{x_2 \mapsto 1, \ldots, x_{i-1} \mapsto 1, x_i \mapsto 0, \ldots, x_{k-2} \mapsto 0\}$$

satisfies G. ■

Observe that G' contains $3(k-2)$ literals, so there is a linear (up to 3 times) blowup.
This can be enhanced to just $\log k$ new literals, by splitting the clause in half and introducing a new variable at each step. That is, write $(\ell_1 \vee \cdots \vee \ell_k)$ as

$$(x_1 \vee \ell_1 \vee \cdots \vee \ell_{\frac{k}{2}}) \wedge (\neg x_1 \vee \ell_{1+\frac{k}{2}} \vee \cdots \vee \ell_k).$$

1.8.2. 2-sat

The *implication graph* (V, E) of a 2-sat formula F with $\mathsf{Vars}(F) = \{p_1, \ldots, p_n\}$ is defined by

$$V = \{p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n\} \text{ and } E = \{(\overline{\ell}_1, \ell_2), (\overline{\ell}_2, \ell_1) : (\ell_1 \vee \ell_2 \in F)\},$$

where $\overline{p} = \neg p$ and $\overline{\neg p} = p$.

Observe that if there is a path from ℓ_1 to ℓ_2, there is a path from $\overline{\ell}_2$ to $\overline{\ell}_1$.
Suppose that for every strongly connected component (scc) $S \subseteq V$ in (V, E), there is a scc S^c, known as the *complementary component*, with exactly the negations of the literals in S.
For each $m \vDash F$, if there is a path from ℓ_1 to ℓ_2, then if $m(\ell_1) = 1$, then $m(\ell_2) = 1$. As a consequence, in a strongly connected component, either all $m(\ell)$ are 1 or all are 0.

The *reduced implication DAG* (V^R, E^R) of an implication graph (V, E) is defined by

$$V^R = \{S : S \text{ is a scc in } (V, E)\} \text{ and } E^R = \{(S, S') : \exists \ell \in S, \ell' \in S' \text{ such that } (\ell, \ell') \in E\}.$$

Theorem 1.12. A 2-sat formula F is unsat iff there is a scc S in the implication graph such that $\{p, \neg p\} \subseteq S$ for some p.

Suppose there is no such S. Define a model as follows. Assign all the literals of an unassigned scc 1 if all its children (the vertices it has an edge to in the reduced implication DAG) are assigned 1. Consequently, assign its complement 0 (this uses the fact that there is no such S). One of the directions of the result follows. There is no violation at any point – this can easily be proved by induction (or the well-ordering principle).
For the other direction, if there is a path from p to $\neg p$, we get a contradiction quite easily, resulting in unsatisfiability.

It is seen from this theorem that a 2-sat formula can be solved in linear time.

1.8.3. XOR-sat and Horn Clauses

A formula is *XOR-sat* if it is a conjunction of xors of literals. Since xors are negations of equalities, we can eliminate via substitution.
For a variable p and formulas F, G, $(p \oplus G) \wedge F$ and $F[\neg G/p]$ are equisatisfiable. This allows us to do substitution repeatedly after determine satisfiability in polynomial time.

A *Horn clause* is a cluase of the form $\neg p_1 \lor \cdots \lor \neg p_n \lor q$, where $p_1, \ldots, p_n \in \mathsf{Vars}$ (where $n \geq 0$) and $q \in \mathsf{Vars} \cup \{\bot\}$. A *Horn formula* is a conjunction of Horn clauses. The clauses with $\bot$ literals are called *goal clauses* and the others are called *implication clauses*. Note that an horn clause is equivalent to

$$p_1 \land \cdots \land p_n \implies p.$$

Note that the trivial model that assigns 0 to everything need not always work because n can be 0.
Start with a model all 0. Pick an implication clause not satisfied yet. Set the right to true. If it causes a problem at some goal clause, then return unsat.

1.9. Lecture 9

1.9.1. Resolution Proof Systems

For a clause C and a literal ℓ, $\ell \cup C$ denotes $\{\ell\} \cup C$.
If we start with a set of CNF formulas, how many rules do we need? It turns out we only need two - Assumption and Resolution.
When writing rules, we usually omit Σ if it is clear from context. So for example, we can apply Assumption as

$$\frac{p \lor C \qquad \neg p \lor C}{C \lor D}$$

The two on the top are called *antecedent* and the bottom is called the *resolvent*. p is called the *pivot* here.

It is possible in some cases that we have multiple choices for the pivot. For example, consider

$$\frac{p \lor q \lor r \qquad \neg p \lor \neg q \lor r}{q \lor \neg q \lor r}$$

and

$$\frac{p \lor q \lor r \qquad \neg p \lor \neg q \lor r}{p \lor \neg p \lor r}$$

In one case, we take p as the pivot and in the other, we take q. However, if we have a choice in the pivot, then it will be like the above, wherein both p and $\neg p$ are in the resolvent. Therefore, any non-trivial resolution will have a unique pivot.
The resolution proof method takes a set of clauses Σ and produces a forest of clauses as a proof. The goal of the proof method is to find the empty clause (which signifies an inconsistency).

For example, consider the context of the derivation to be $\Sigma = \{p \lor q, \neg p \lor q, \neg q \lor r, \neg r\}$. Then

$$\dfrac{\dfrac{\dfrac{p \lor q \qquad \neg p \lor q}{q} \qquad \neg q \lor r}{r} \qquad \neg r}{\bot}$$

The lower we go, the more the *depth* increases.
Recall that formal proof systems do not explicitly refer to $\bot$, they encode $\bot$ by $F \land \neg F$ for some formula F. Even in the above system for example, we actually get $\Sigma \vdash \neg r$ and $\Sigma \vdash r$.
Therefore, if a resolution proof system can derive $\Sigma \vdash \bot$, Σ is unsatisfiable (due to the soundness of our system).
Say we want to prove $\Sigma \vdash F$. We convert $\neg F \land \bigwedge \Sigma$ into a new set of clauses Σ' and apply the resolution proof method on this instead. If we derive $\bot$, $\Sigma \vdash F$ is derivable.

1.9.2. Optimizations

How do we find resolution proofs? There are some issues in implementation.
A proof method implicitly defines a non-deterministic proof search algorithm. To implement such an algorithm, we should ensure that we are not doing unnecessary work. For now, we only worry about a single rule.

1. For clauses C and D, if $D \subseteq C$ and $\perp$ can be derived using C, then it can be derived using D.
   If $D \subseteq C$, C is said to be *redundant*. For example, $\neg q \vee r$ is redundant in $\{q, \neg q \vee r, r, \neg r\}$ (since $r \subseteq \neg q \vee r$). Shorter clauses help us get to the empty clause faster.

2. If a clause contains both p and $\neg p$, it is valid.
   If a valid clause contributes in deriving $\perp$, the descendants must participate in some resolution step with pivot p. This step is counterproductive (the resolvent is a superset of some antecedent). For example,

$$\frac{p \vee C \qquad \neg p \vee p \vee D}{p \vee C \vee D}$$

3. If a literal occurs in a CNF formula and its negation does not, it is said to be a *pure literal*. The removal of clauses containing pure literals in a CNF preserves satisfiability.

4. If ℓ occurs in a resolution proof, $\neg \ell$ can be removed from every clause.

1.10. Lecture 10

1.10.1. Completeness of the resolution proof system

If Σ is unsatisfiable, are we guaranteed to derive $\Sigma \vdash \perp$ via resolution?

We define the set $\mathsf{Res}^n(\Sigma)$ of clauses that are derivable via resolution proofs of depth n from the set of clauses Σ. So $\mathsf{Res}^0(\Sigma) \coloneqq \Sigma$ and

$$\mathsf{Res}^{n+1}(\Sigma) \coloneqq \mathsf{Res}^n(\Sigma) \cup \{C : C \text{ is a resolvent of clauses } C_1, C_2 \in \mathsf{Res}^n(\Sigma)\}$$

For example,

$$\mathsf{Res}^1(\{p \vee q, \neg p \vee q, \neg q \vee r, \neg r\}) = \Sigma \cup \{q, p \vee r, \neg p \vee r, \neg q\}.$$

Since there are only finitely many variables in Σ, we can only derive finitely many clauses.

Definition 1.20. Let Σ be a finite set of clauses. There exists an m such that

$$\mathsf{Res}^{m+1}(\Sigma) = \mathsf{Res}^m(\Sigma).$$

We define $\mathsf{Res}^*(\Sigma) \coloneqq \mathsf{Res}^m(\Sigma)$.

Theorem 1.13. If a finite set of clauses Σ is unsatisfiable, $\perp \in \mathsf{Res}^*(\Sigma)$.

Proof. We prove this by induction over the cardinality of Σ. We assume there are no tautology classes in Σ.
Let $\Sigma = \{p\}$. If it is unsat, then $\{p, \neg p\} \subseteq \Sigma$. The derivation to get $\perp$ is obvious.
Suppose it holds for all formulas containing variables $p_1, \ldots, p_n$. Let Σ be an unsat set of clauses with variables $p_1, \ldots, p_n, p$. Let Σ_0 and Σ_1 be the set of clauses from Σ that have p and $\neg p$ respectively. Let Σ_* be the remaining clauses. Clearly, $\Sigma = \Sigma_0 \wedge \Sigma_1 \wedge \Sigma_*$. Now, define $\Sigma_0' = \{C \setminus \{p\} : C \in \Sigma_0\}$. Similarly define Σ_1' by removing $\neg p$.
It may be shown that $\Sigma_0' \models \Sigma_0$ and $\Sigma_1' \models \Sigma_1$.

Consider the formula $(\Sigma_0' \wedge \Sigma_*) \vee (\Sigma_1' \wedge \Sigma_*)$. Clearly, p is not in this formula.
We claim that if this formula is sat, so is Σ. Let m satisfy this formula. Then $m \models \Sigma_*$. If $m \models (\Sigma_1' \wedge \Sigma_*)$, then since all the clauses of Σ_0 have p, $m[p \mapsto 1] \models \Sigma_0$ and since Σ_1' and Σ_* don't have p, $m[p \mapsto 1] \models \Sigma_1'$ and $m[p \mapsto 1] \models \Sigma_*$. Since $\Sigma_1' \models \Sigma_1$, $m[p \mapsto 1] \models \Sigma_1$. Therefore, $m[\mapsto 1] \models (\Sigma_0 \wedge \Sigma_1 \wedge \Sigma_*)$. Similarly, if $m \models (\Sigma_0' \wedge \Sigma_*)$,

then $m[p \mapsto 0] \vDash (\Sigma_0 \wedge \Sigma_1 \wedge \Sigma_*)$.
Now, since Σ is unsat, so is $(\Sigma_0' \wedge \Sigma_*) \vee (\Sigma_1' \wedge \Sigma_*)$.

Since they don't have p, $\perp \in \mathsf{Res}^*(\Sigma_0' \wedge \Sigma_*)$ and $\perp \in \mathsf{Res}^*(\Sigma_1' \wedge \Sigma_*)$. Choose a derivation of $\perp$ from both.
If $\perp$ was derived using only clauses from Σ_* from either proof. Then we are done.
If Σ_0' and Σ_1' are involved in each of the derivations, then $p \in \mathsf{Res}^*(\Sigma_0 \wedge \Sigma_*)$ and $\neg p \in \mathsf{Res}^*(\Sigma_1 \wedge \Sigma_*)$ ("add" p and $\neg p$ to each of the clauses that involve Σ_0' and Σ_1' respectively). Again, the required follows, thus proving the theorem. ■

1.10.2. Compactness and Implication

Let Σ be a finite set of formulas and F be a formula. The following are equivalent.

(i) $\Sigma \vdash F$.

(ii) $\varnothing \in \mathsf{Res}^*(\Sigma')$, where Σ' is the CNF of $\bigwedge \Sigma \wedge \neg F$.

(iii) $\Sigma \vDash F$.

(i) to (iii) says that proof rules are sound. (ii) to (i) says that a proof can be generated. (iii) to (ii) is what we showed in the previous theorem.

But what do we do in the case where Σ is infinite?
Suppose $\Sigma'' \vDash F$ and Σ'' is infinite. We shall show that this implies that this is truee iff there is some finite $\Sigma \subseteq \Sigma''$ such that $\Sigma \vDash F$.

Theorem 1.14. Let S be an infinite set of finite binary string. There exists an infinite string w such that

$$|\{w' \in S : w_n \text{ is a prefix of } w'\}| = \infty$$

for all n where w_n is a prefix of w of length n.

Proof. We inductively consutrct w and keep shrinking S. Initially, $w = \epsilon$. Let $S_0 = \{u \in S : u \text{ starts with } 0\}$, $S_1 = \{u \in S : u \text{ starts with } 1\}$, and $S_\epsilon = S \cap \{w\}$. Clearly, $S = S_\epsilon \cup S_0 \cup S_1$. At least one of S_0 and S_1 is infinite. Suppose S_0 is infinite. Then "expand" w to 0 and restrict to $S = S_0$.
Suppose we have w of length n and w is a prefix of all string in S. Again, let $S_0 = \{u \in S : u \text{ has 0 at the } n\text{th position}\}$, $S_1 = \{u \in S : u \text{ has 1 at the } n\text{th position}\}$, and $S_\epsilon = S \cap \{w\}$. If S_0 is infinite, expand w to $w0$ and $S = S_0$. Otherwise, $w = w1$ and $S = S_1$.
The result follows. ■

Theorem 1.15. A set Σ of formulas is satisfiable iff every finite subset of Σ is satisfiable.

Proof. The forward direction is trivial.
Order the formulas of Σ as $\Sigma = \{F_1, F_2, \ldots\}$. Let $\{p_1, p_2, \ldots\}$ be an ordered list of variables from $\mathsf{Vars}(\Sigma)$ such that we first write formulas in $\mathsf{Vars}(F_1)$, then $\mathsf{Vars}(F_2) \setminus \mathsf{Vars}(F_1)$, and so on.
We have models m_n such that $m_n \vDash \bigwedge_{i=1}^n F_i$.
We assume $m_n : \mathsf{Vars}\left(\bigwedge_{i=1}^n F_i\right) \to \mathcal{B}$. We can think of the (m_n) as finite binary strings since the variables are ordered $p_1, p_2, \ldots$ and m_n assigns values to the first k variables as a string. let $S = \{m_n \text{ as a string} : n > 0\}$. Let m be the infinite string as mentioned in the previous theorem.
If we then interpret m as a model, then $m \vDash \Sigma$. Consider a formula $F_n \in \Sigma$ and suppose there are k variables up to F_n. Let m' be the prefix of length k of m. There must be $m_j \in S$ such that m' is a prefix of m_j and $j > n$. Since m_j satisfies the conjunction of the n formulas, $m_j \vDash F_n$. Therefore, $m' \vDash F_n$ and $m \vDash F_n$. ■

If we can enumerate a set using some algorithm, it is said to be *effectively enumerable*.

A yes/no problem is *semi-decidable* if we have an algorithm for only one side of the problem is decidable (either yes or no).

Theorem 1.16. If Σ is effectively enumerable, $\Sigma \vDash F$ is semi-decidable.

Proof. If $\Sigma \vDash F$, there is a finite set $\Sigma_0 \subseteq \Sigma$ such that $\Sigma_0 \vDash F$. Since Σ is effectively enumerable, let $G_1, G_2, \ldots$ be an enumeration of it. Let $S_n = \{G_1, \ldots, G_n\}$.
There is then some k such that $S_k \vDash F$. We may enumerate S_n and check if $S_n \vDash F$ (which is decidable by virtue of truth tables). Therefore, if $\Sigma \vDash F$, we will eventually say yes. So it is decidable if the answer is yes, but not if no. ■

§2. Exercises

Exercise 2.0.1. Suppose we have n variables in a 2-sat problem. What is the maximal number of clauses in the formula such that the formula is satisfiable?

Solution

Observe that it is possible to have $3n^2$ clauses. Indeed, consider the set of clauses

$$\{\{p, \neg q\} : p, q \in \mathsf{Vars}\} \cup \{\{p, q\} : p, q \in \mathsf{Vars}\}$$

which is satisfiable by the model that takes 1 everywhere.
We claim that this is the maximal number of clauses. Indeed, let us have an implication graph with $2k \geq 2$ strongly connected components. Also, we can use the notion of complementary components and let the sccs have $v_1, \ldots, v_k$ vertices (where there are two sccs corresponding to each v_i).
The number of edges is then at most

$$2 \sum_{1 \leq i \leq k} v_i^2 + \sum_{1 \leq i \leq k} v_i^2 + \sum_{1 \leq i < j \leq k} 4 v_i v_j.$$

The first term arises due to complete connectedness within each scc and the second and third terms arise by taking into consideration the maximal number of edges between sccs. The above is equal to

$$3 \left(\sum_{1 \leq i \leq k} v_i \right)^2 - 2 \sum_{1 \leq i < j \leq k} v_i v_j = 3n^2 - 2 \sum_{1 \leq i < j \leq k} v_i v_j \leq 3n^2,$$

thus proving the claim.