Mathematical Logic

Massimo Perego, Omar Masri

Last updated: December 5, 2024

Contents

※ Introduction

this introduction should be redone

Intuitively two Sentences "say the same thing" **iff** they're true in exactly the same circumstances. Starting on logic: from the sentences
Example 1.0.1. This is an example of **syllogism**:

- Every man is mortal
- Socrates is a man

- Therefore, Socrates is mortal

Consequences from some assumptions, which are considered true. The concept is that **if the assumptions are true, then the consequences must also be true**.

Logic doesn't really think about each part of the sentence in a literal sense (we don't care what they "mean" in the real world), but **each sentence has to be formalized in a way that makes the "shape" of the information the most important part**.
Example 1.0.2. Another example:

- All cats have seven legs
- Bob is a cat

- Therefore Bob has seven legs

We can say this because the "shape" of the argument is the same as before, even if rationally it doesn't make sense, it's not "right", but **in a world in which these informations are true then the consequences must be true**.

We don't care about the content, **we only care about the shape and what our clauses imply**.
Example 1.0.3. Yet Another example:

- Every tik is tok
- Tak is tik

- Therefore tak is tok

It doesn't mean shit, but this is true whatever being tik, tak or tok means. We'll eventually want to model pieces of the real world, but logic doesn't necessarily care about that.

Formalizing the shape of these examples:

- First sentence
$$\forall x \quad P(x) \to Q(x)$$
For every individual in the universe, possessing the property $P()$ implies possessing $Q()$.

- Second sentence
$$P(s)$$
- Therefore
$$Q(s)$$

We **formalized** the sentences, but we **lost some information in the process**, we lose the facts that we're talking about Socrates, or cats, we don't know that we're talking about.

Example 1.0.4. Lets Consider another example:

- Every flower is perfumed
- The rose is perfumed

- Therefore the rose is a flower?

Is this syllogism correct? Let's formalize it to find out.

$$\forall x \quad P(x) \to Q(x)$$
$$Q(s)$$

Therefore $P(s)$?

This obviously **doesn't hold**, our first predicate doesn't automatically works bidirectionally, we're **reversing our implication**. With our assumptions there's no "proof" that $P(s)$ is true.

Example 1.0.5. Final example:

- If it rains I take my umbrella

Which one is the **equivalent?**

1. If it doesn't rain I don't take my umbrella
2. If I don't take my umbrella then it doesn't rain
3. If I take the umbrella then it rains
4. Either it doesn't rain or I take the umbrella
5. It rains only if I take the umbrella
6. It rains if I take the umbrella
7. It rains if and only if I take the umbrella
8. None of the above

To answer we need to formalize what we mean by equivalent: for logicians it means "given a world in which its true then the other is true and given a world in which it's false then the other is false".

With this definition then the **second** one is **true**:

$$P(x) \to Q(x) \text{ then } \neg Q(x) \to \neg P(x)$$

The first one is a simple negation, it can't be true as well.

The third one is the opposite of the original sentence.

There's at least two types of *or* in a natural language, usually it's exclusive. The **fourth sentence is correct if the *or* is inclusive**, if it's a *xor* it can't be.

Five is also a **correct** translation, once we agree on the meaning of words like "if", "only if", ...; It basically means that the first part (rain) can or can not happen based on the "umbrella" fact, which is the same as the original.
We're speaking in both cases about a necessary and sufficient condition for rain. Taking the umbrella is a necessary condition for rain, it doesn't cause the rain. There's no causation between clauses.

Six and seven are false, since five is true.

Equivalence: What does "saying the same thing" means in logic terms?
Two sentences **say the same thing if and only if (iff) they are true in exactly the same circumstances**, a.k.a. "possible world", a.k.a. "truth evaluation".

If *it rains* then *I take my umbrella*: the cursive ones are stand alone sentences, each one is an **atomic sentence** (in logic terms), they cannot be further disassembled.

So, **formalizing** the sentence

- it rains $= P$
- I take my umbrella $= Q$
- if/then $=\to$

$$P \to Q$$

There can be many (infinite) world in which "it rains", i.e. P, is true and another set of worlds in which "I take my umbrella", i.e. Q, is true. These sets can (will) intersect.

Imagining something along the lines of a Venn diagram (I don't care enough to actually draw it), there will be the following regions:

- P: worlds in which it only rains
- Q: worlds in which I only take my umbrella
- $P \cap Q$: worlds in which it rains and I take my umbrella
- $\neg(P \vee Q)$: worlds in which it doesn't rain and I don't take my umbrella (outside all)

For each region we have

- it rains but I don't take my umbrella (inside the P region only) $\implies$ for the worlds in here the preposition is false
- it doesn't rain but I take my umbrella (Q region) $\implies$ for the worlds in here the preposition is true
- inside both regions, it rains and I take my umbrella $\implies$ for the worlds here the preposition is true
- outside both regions, doesn't rain, don't take my umbrella $\implies$ for the worlds here the preposition is true

Now we have a sort of "image" on which to evaluate if other sentences are equivalent. **If another preposition gives us the same image**, i.e. the same truth values in the same regions of the diagram, **it's equivalent to our original**.

Considering our earlier example and taking sentence #4:

$$(\neg P) \vee Q$$

- $\neg P$ is true when P is not (duh)
- $P \vee Q$ is true inside the P, Q and intersection region between them

So, combining these two gives us that, $(\neg P) \vee Q$ is true

- inside Q region
- inside intersection between P and Q
- outside everything

So it's the same as our initial example $P \to Q$, they give the same "picture", so they say the same thing, they are equivalent. **They are true in the same circumstances**. Each world is false (or true), i.e. has the same value, in both cases.

Another case, sentence #2:

$$\neg Q \to \neg P$$

gives the same picture, while sentence #1

$$\neg P \to \neg Q$$

does not (obviously, they're opposite to one another, they can't be both true).

Considering sentences #5

$$Q \leftarrow P \implies P \to Q$$

and #6

$$Q \to P$$

#6 can be rephrased in #6′ as: If I take my umbrella it rains. Doesn't sound too correct, does it?
While in sentence #5 "only if" goes the other way in respect to "if".

Basically:

- **if** goes **one way**
- **only if** goes the **other way**
- **if and only if (iff)** goes **both ways**

Inside

$$P \to Q : true$$

It means that:

- P is **sufficient** for Q to be true (if there's P there must be Q)
- Q is **necessary** for P to be true (there can't be P without Q)

There can be no case in which P (it rains) is true and Q isn't (I take my umbrella).

When is our preposition true? "$P \to Q$ is true" can be read as:

"When P is true then Q is true"

or

"Each single time P is true then Q is true (in every possible world)"

※ Introduction to Formulas

A **"Sentence"** (or **proposition**) in natural language is a **grammaticarly correct sequence of words such that it makes sense to ask whether** (in the given circumstance) **it is true or false**.
Example 2.0.1. Let's make some examples of propositions

- It rains ✓
- If it rains, I stay at home ✓
- What's the time? × Questions are not sentences.
- $2+2=4$ ✓
- $2+2=5$ ✓ It's wrong, but still a sentence.

2.1 Meaning: Denotation vs Connotation

Consider:

- 4
- 2^2
- $6-2$
- $5+1$

They're **not sentences** because they **can't be true or false**, they just are. But all these expressions **mean the same thing**, they all **denote** the natural number 4 (apart from the last one). This is the denotation of these expressions. They are all expressions that describe distinct ways to obtain 4. The **denotation** is the number 4, **all the other informations given by the expressions are connotations**.

For each given an expression E we have the denotation of E and the connotation(s) of E.

A **denotation** is just a **name** (a pointer), **referring to a uniquely determined object** of discourse.

The **connotations** are the **names with all the actual information they contain**. A name can be a complex thing and as such can contain informations.

We'll primarily focus on denotations, as connotations are more challenging to handle, and denotations possess the following useful property.
Property 2.1.1 (Invariance under substitutions)**.** Two different ways to connote the same denoted object can be substituted as they mean the same thing. For example we can exchange 4 and $2+2$ because they're different strings but evaluate to the same object.

Ok but what is the denotation of sentences in standard arithmetic?

- $4 = pred(5)$ $true$
- $4 = succ(5)$ $false$
- $4 = 3 + 1$ $true$
- $4 = 4$ $true$
- $2 = 2$ $true$
- $4 = 6$ $false$

The **denotation of a sentence**, in a given possible world, is just **one of two truth values**, $true$ or $false$. **Each sentence is evaluated to a precise truth value**.

2.2 Formalization

What is propositional logic about? Propositions (a.k.a. sentences) and their semantics (denotation, their truth values) in every possible circumstance.

To formalize sentences we distinguish among **two types of sentences**:

- **Atomic sentences**: Simple sentences (e.g., "It rains," "Paul runs," "I stay at home") that cannot be simplified further.
- **Complex sentences**: Formed by combining atomic sentences with connectives (It rains and Paul runs, Paul runs or I stay at home, ...). These can be arbitrarily complex.

The **formalization** of these types of sentences is:

- **Atomic sentences** $\rightarrow$ **Atomic formulas**, they **become symbols** (p, q, r, p_1, ...).
  We fix a (countably) infinite set of symbols $\mathcal{L}$ as out set of atomic formulas, $\mathcal{L}$ is called "propositional language" and its elements are called "atomic formulas" or "propositional letters/variables". In every possible world, every propositional letter can either be true or false.

- **Complex sentences**: Fix a propositional language $\mathcal{L}$ and the **set of propositional formulas on this language**.
  $\mathcal{F}_\mathcal{L}$ is defined as follows (3 ways):
  1. The smallest set such that:
     - $\forall p \in \mathcal{L}$ then $p \in \mathcal{F}_\mathcal{L}$ $(\mathcal{L} \subseteq \mathcal{F}_\mathcal{L})$
     - $\forall A, B \in \mathcal{F}_\mathcal{L}$ then $(A \cap B)$, $(A \cup B)$, $(A \rightarrow B)$, $(\neg A) \in \mathcal{F}_\mathcal{L}$
  2. $\mathcal{F}_\mathcal{L}$ is the **intersection** of all sets $\mathcal{X}$ such that:
     - $\mathcal{L} \subseteq \mathcal{X}$
     - $\forall A, B \in \mathcal{X}$ then $(A \cap B)$, $(A \cup B)$, $(A \rightarrow B)$, $(\neg A) \in \mathcal{X}$
  3. Inductive definition: $\mathcal{F}_\mathcal{L}$ is the set such that the following properties hold:
     (Base) $\mathcal{L} \subseteq \mathcal{F}_\mathcal{L}$
     (Inductive Step) If $A, B \in \mathcal{F}_\mathcal{L}$ then $(A \cap B)$, $(A \cup B)$, $(A \rightarrow B)$, $(\neg A) \in \mathcal{F}_\mathcal{L}$
     (3) Nothing else belongs to $\mathcal{F}_\mathcal{L}$

  The difference between first and last one is that in the former we specify "smallest set", while in the last it's a property.

Example 2.2.1. given $p, q, r \in \mathcal{L}$, then

- $((p \wedge q) \rightarrow r) \in \mathcal{F}_\mathcal{L}$ this is a valid formula and as such is in $\mathcal{F}_\mathcal{L}$
- $p \vee \wedge(q \wedge r) \notin \mathcal{F}_\mathcal{L}$ this is a string but it's not a formula

We must be able to identify formulas from meaningless strings. To achieve this, we require a

certificate, some type of proof, that confirms whether a given string is a formula.

It is important to note that the set of connectives used in any definition is chosen arbitrarily; there can be many, few, or even infinitely many connectives. The formalization of complex sentences depends on the specific connectives under consideration.

2.3 Certifying formulas

We need to **produce a certificate that a string is a formula**, a kind of proof that something is a formula. We don't need to prove that something is NOT a formula since we indirectly get that from the fact that there's no certificate for such string.

2.3.1 $\mathcal{L}$-construction

Given a word $w \in (\{\wedge, \vee, \neg, \rightarrow,), (\} \cup \mathcal{L})^*$, an $\mathcal{L}$-construction is a way to **show that**, w **is actually a formula**, i.e. $w \in \mathcal{F}_{\mathcal{L}}$.

more formally an $\mathcal{L}$-construction of w is a sequence finetly many strings $w_1, w_2, ..., w_u$ such that:

- $w_u = w$
- $\forall i = 1, 2, ..., u$, either.
  - $w_i \in L$
  - $\exists j < i$ such that $w_i = (\neg w_j)$
  - $\exists j, k < i$ such that
    $w_i = (w_j \wedge w_k)$
    $w_i = (w_j \vee w_k)$
    $w_i = (w_j \rightarrow w_k)$

The string must be the negation of something else, the conjunction of something else or the disjunction of something else, implication, ecc.; it must be made of something else which is simpler.

Example 2.3.1. Let's show more than one $\mathcal{L}$-construction that proves $(p \wedge q) \rightarrow r \in \mathcal{F}_{\mathcal{L}}$

- $p, \quad q, \quad r, \quad (p \wedge q), \quad ((p \wedge q) \rightarrow r)$
- $r, \quad q, \quad p, \quad p, \quad (p \rightarrow p), \quad (p \wedge q), \quad ((p \wedge q) \rightarrow r)$

These aren't the only $\mathcal{L}$-construction possible, there are infinitely many possible ones (we can add useless stuff and even repetitions).

But we can have a **shortest** $\mathcal{L}$-construction, its easy to prove that such $\mathcal{L}$-construction will be as long as the number of letters + the number of connectives used in the formula we need to certify.

2.3.2 Unique readability

Directly from the definition of $\mathcal{L}$-constructions comes another property: **unique readability**. From the definition of $\mathcal{L}$-construction, it can be observed that if $w \in \mathcal{F}_{\mathcal{L}}$ then **exactly one** of the following cases **holds**

- $w \in \mathcal{L}$
- or there is $v \in \mathcal{F}_{\mathcal{L}}$ such that $w = (\neg v)$
- or there are $v_1, v_2 \in \mathcal{F}_{\mathcal{L}}$ such that $w = (v_1 \wedge v_2)$
- or there are $v_1, v_2 \in \mathcal{F}_{\mathcal{L}}$ such that $w = (v_1 \vee v_2)$

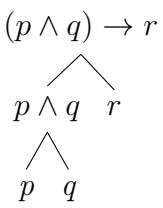
- or there are $v_1, v_2 \in \mathcal{F}_\mathcal{L}$ such that $w = (v_1 \to v_2)$

Essentially, either w it's not decomposable, or it's decomposable in a unique way.

Not every formal language possesses the psroperty of unique readability. A simple example of a formal language lacking this property is the case where $w \in \Sigma^*$ and $w = v_1 \cdot v_2$ (the concatenation of strings). For instance, consider $w =$ hello. This string can be decomposed in 6 different ways of the form $w = v_1 \cdot v_2$:

- $\epsilon, hello$
- $h, ello$
- he, llo
- hel, lo
- $hell, o$
- $hello, \epsilon$

The unique readability of formulas grants us additional forms of certification for "Formulicity" Thanks to this property, we can construct a **unique parsing tree** for any given formula.
Example 2.3.2. Lets draw the parsing tree of $(p \wedge q) \to r$:



For a given string $w \in (\{\wedge, \vee, \neg, \to,), (\} \cup \mathcal{L})^*$ if we have a single parsing tree, it's for sure a formula.

2.3.3 Structural induction principle

Since we can **define $\mathcal{F}_\mathcal{L}$ as an inductive set** (third definition), it comes with its own **Induction principle**.
Definition 2.3.3. Let $\mathcal{P}$ be some property that can be asked of formulas (note that $\mathcal{P}$ may hold or not for any given $\mathcal{F} \in \mathcal{F}_\mathcal{L}$).

This property $\mathcal{P}$ holds for all formulas $\mathcal{F} \in \mathcal{F}_\mathcal{L}$ iff:

1. **base**: $\mathcal{P}$ holds for all propositional letters $p \in \mathcal{L}$
2. **step(s)**: if $\mathcal{P}$ holds for generic formulas $A, B \in \mathcal{F}_\mathcal{L}$ then $\mathcal{P}$ holds for all the ways to build new formulas starting from A and B: $(\neg A), (A \wedge B), (A \vee B), (A \to B)$

Proof.

- Let $\mathcal{P}$ be a property satisfying **base** and **step(s)**.
- Let $\mathcal{I} = \{\mathcal{F} \in \mathcal{F}_\mathcal{L} : \mathcal{P} \text{ holds for } \mathcal{F}\}$ (the set of formulas for which $\mathcal{P}$ holds).

We now need to show that $\mathcal{F}_\mathcal{L} = \mathcal{I}$ (the set of formulas in which the property holds coincides with the complete set of formulas). To do that we need to show that:

1. $\mathcal{F}_\mathcal{L} \subseteq \mathcal{I}$
2. $\mathcal{I} \subseteq \mathcal{F}_\mathcal{L}$

2. is trivial since $\mathcal{I}$ is defined as a subset of $\mathcal{F}_\mathcal{L}$. We only need to prove that all the formulas are in $\mathcal{I}$.

Since $\mathcal{P}$ satisfies the **base** case, it follows that $\mathcal{P}$ holds for all $p \in \mathcal{L}$. Therefore, $\mathcal{L} \subseteq \mathcal{I}$.

If $A, B \in \mathcal{I}$, then $\mathcal{P}$ holds for A and B. By **step(s)**, $\mathcal{P}$ also holds for $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, and $(A \to B)$. Consequently, the same constructions we defined for $\mathcal{F}_\mathcal{L}$ belong to $\mathcal{I}$.

Thus, each formula $\mathcal{F}$ belongs to $\mathcal{I}$, which implies $\mathcal{F}_\mathcal{L} \subseteq \mathcal{I}$. □

2.4 Semantics of propositional logic

Classical propositional logic is based on a couple of **underlying design ideas**:

- **Bivalence principle**: a sentence (formula) in every possible universe **is either true or false**.
- **Truth-functionality principle**: (also called compositionality, extensionality) the **truth value of a sentence** (a formula) in a given possible universe **only** depends on the truth values of the sentences composing it and the fixed meaning of connectives.

An easy example of a non truth-functional realm is probability we can prove this with a simple example

Example 2.4.1. given $A :=$ Tomorrow it will rain and $B :=$ France is going to win the next world cup and given the probabilities of those two events being $v(A) = v(B) = 0.5$ then by truth functionality $v(A \to B) = v(\frac{1}{2} \to \frac{1}{2}) = v(A \to A)$ but this makes no sense since the probability of $v(A \to A)$ its obviously 1.

For instance, take conjunction

$$A, B \in \mathcal{F}_\mathcal{L} \qquad A \wedge B$$

We want to evaluate the value in a given possible world. By bivalence we know that this is either true or false. By truth-functionality this value depends only on:

- whether A is true or false
- whether B is true or false

The notion of "possible world": it's a **function** from the set of propositional letters to the set of truth values

$$v : \mathcal{L} \mapsto \{0, 1\}$$

It assigns a truth value to each of the propositional letters. The set of all possible worlds is the set of all possible truth evaluations.

Now Let's formalize these notions, by truth functionality lets fix the semantics of our connectives

$$\tilde{v}(A \wedge B) := I_\wedge(\tilde{v}(A), \tilde{v}(B))$$
$$I_\wedge : \{0, 1\}^2 \mapsto \{0, 1\} = a \cdot b = \min\{a, b\}$$

a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

$$I_\vee : \{0, 1\}^2 \mapsto \{0, 1\} = a + b = \max\{a, b\}$$

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

$$I_{\neg} : \{0,1\} \mapsto \{0,1\} = 1 - a$$

a	$\neg a$
0	1
1	0

$$I_{\rightarrow} : \{0,1\}^2 \mapsto \{0,1\} = \max\{1 - a, b\}$$

a	b	$a \rightarrow b$
0	0	1
0	1	1
1	0	0
1	1	1

$$I_{\leftrightarrow} : \{0,1\}^2 \mapsto \{0,1\} = ((a + b)\ mod\ 2) + 1$$

a	b	$a \leftrightarrow b$
0	0	1
0	1	0
1	0	0
1	1	1

$$I_{\oplus} : \{0,1\}^2 \mapsto \{0,1\} = ((a + b)\ mod\ 2)$$

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Definition 2.4.2. A **truth evaluation is an arbitrarily chosen map**

$$v : \mathcal{L} \mapsto \{0,1\}$$

Definition 2.4.3. the canonical extension $\tilde{v} : \mathcal{F}_{\mathcal{L}} \mapsto \{0,1\}$ of $v : \mathcal{L} \mapsto \{0,1\}$ is defined as follows:

- if $F \in \mathcal{L}$ then $\tilde{v}(F) := v(F)$
- if $A, B \in \mathcal{F}_{\mathcal{L}}$ then
  $\tilde{v}(A \wedge B) := I_{\wedge}(\tilde{v}(A), \tilde{v}(B))$
  $\tilde{v}(A \vee B) := I_{\vee}(\tilde{v}(A), \tilde{v}(B))$
  $\tilde{v}(A \rightarrow B) := I_{\rightarrow}(\tilde{v}(A), \tilde{v}(B))$
  $\tilde{v}(\neg A) := I_{\neg}(\tilde{v}(A))$

2.4.1 Tautologies and other definitions

Definition 2.4.4 (Tautology)**.** a formula $F \in \mathcal{F}_{\mathcal{L}}$ is a **Tautology iff** $\forall v : \mathcal{L} \mapsto \{0,1\}$, $\tilde{v}(F) = 1$, i.e. it's always true for all assignment.
Definition 2.4.5 (Satisfiable Formula)**.** a formula $F \in \mathcal{F}_{\mathcal{L}}$ is **Satisfiable iff** $\exists v : \mathcal{L} \mapsto \{0,1\}$, $\tilde{v}(F) = 1$, i.e. there's at least one assignment that makes it true.
Definition 2.4.6. a formula $F \in \mathcal{F}_{\mathcal{L}}$ is (**contradiction**, **unsatisfiable**, **refutable**): **iff** $\forall v : \mathcal{L} \mapsto \{0,1\}$, $\tilde{v}(F) = 0$, i.e. it's never true for all assignment.
Fact 2.4.1. Let $F \in \mathcal{F}_{\mathcal{L}}$ then F is a tautology **iff**

$$\neg F \text{ is a contradiction}$$

Proof. F is a tautology **iff** (by definition of tautology) $\forall v : \mathcal{L} \mapsto \{0,1\}$, $\tilde{v}(F) = 1$. But this by defintion of (truth table) $I_{\neg}$ means that $\tilde{v}(\neg F) = 0$ for all $v : \mathcal{L} \mapsto \{0,1\}$, and thus by definition of contradiction $\neg F$ is unsatisfiable. □

To **prove** that the **implication is correct** the way it is:

2.4.2 Digression on combinatorics

$$|\{F : \{0,1\}^n \mapsto \{0,1\}\}| = 2^{2^n}$$

Which is a specific version of a more general formula: with A and B as finite sets

$$|\{F : A \mapsto B\}| = |B|^{|A|}$$

Intuitively this formula can be explained by the fact that there are $|B|$ options for each element of A. Let $\mathcal{S}$ be a set of n many elements. There are 2^n possible subsets of $\mathcal{S}$, we can use the aforementioned formula to prove this

$$2^n = |\mathcal{P}(\mathcal{S})| = 2^{|\mathcal{S}|} = |\{F : \mathcal{S} \mapsto \{0,1\}\}|$$

Note that we are counting all characteristic functions on $\mathcal{S}$, where a characteristic function is:

$$X_A : \mathcal{S} \mapsto \{0,1\} \quad \begin{cases} 1 \text{ if } x \in A \\ 0 \text{ if } x \notin A \end{cases}$$

Remember that characteristic functions are essentially subsets; there is an isomorphism between characteristic functions and subsets.
Example 2.4.7. Let's make an example to justify the interpretation of implications.

"If a natural number is divisible by 4 then it is divisible by 2"

$$n = 4k \implies n = (2 \cdot 2)k \implies 2 \cdot (2k)$$

Here the only assumption is that n is divisible by 4, so we are skipping over the case where $n \neq 4k$. However, let us now examine all cases, even when this assumption does not hold, keeping in mind that the proposition is obviously true over all numbers.

"If 4 is divisible by 4, then 4 is divisible by 2"
$$true \to true = true.$$

"If 3 is divisible by 4, then 3 is divisible by 2"
$$false \to false = true.$$

"If 6 is divisible by 4 then 6 is divisible by 2".
$$false \to true = true.$$

2.4.3 Assignment Encoding

We want to demonstrate how it's possible to **encode** the **infinite information** given by a general truth value assignment on all letters of $\mathcal{L}$ in **finite time**.

Lemma 2.4.8. Let $F \in \mathcal{F}_\mathcal{L}$ and let $v, v' : \mathcal{L} \mapsto \{0,1\}$ be two (possibly) distinct truth value assignments. Then: if $v(p) = v'(p)$ for all $p \in \mathcal{L}$ letters **actually occurring** in F (so occurs in every $\mathcal{L}$-construction of F) then $\tilde{v}(F) = \tilde{v}'(F)$. The only relevant part of an assignment is the **letters actually occurring**, we don't care about all the infinite non-occurring letters.

Proof. Structural induction of F

Base: If $F = p$, $p \in \mathcal{L}$ then p occurs in F, trivially then by hypothesis $v(p) = v'(p)$, then by definition of $\tilde{v}$, $\tilde{v}(p) = \tilde{v}'(p)$.

Step(s):

- If $F = (\neg A)$ for some $A \in \mathcal{F}_\mathcal{L}$, then by induction hypothesis (I.H.) $\tilde{v}(A) = \tilde{v}'(A)$. Since each $p \in \mathcal{L}$ occurring in F occurs in A too, by definition of $I_\neg$:

$$v(F) = \tilde{v}(\neg A) \overset{\text{def}}{=} \neg\tilde{v}(A) \overset{\text{I.H.}}{=} \neg\tilde{v}'(A) \overset{\text{def}}{=} \tilde{v}'(\neg A) = v'(F)$$

- If $F = (A \wedge B)$ for some $A, B \in \mathcal{F}_\mathcal{L}$, then each $p \in \mathcal{L}$ occurring in A occurs in F too and each $p \in \mathcal{L}$ occurring in B occurs in F too. So we can apply the I.H. $\tilde{v}(A) = \tilde{v}'(A)$ and $\tilde{v}(B) = \tilde{v}'(B)$. By definition of $I_\wedge$:

shouldn't these two I.H. have different assignments? TOASK

$$v(F) = \tilde{v}(A \wedge B) \overset{\text{def}}{=} I_\wedge(\tilde{v}(A), \tilde{v}(B)) \overset{\text{I.H.}}{=} I_\wedge(\tilde{v}'(A), \tilde{v}'(B)) \overset{\text{def}}{=} \tilde{v}'(A \wedge B) = v'(F)$$

- If $F = (A \vee B)$ for some $A, B \in \mathcal{F}_\mathcal{L}$, then each $p \in \mathcal{L}$ occurring in A occurs in F too and each $p \in \mathcal{L}$ occurring in B occurs in F too. So we can apply the I.H. $\tilde{v}(A) = \tilde{v}'(A)$ and $\tilde{v}(B) = \tilde{v}'(B)$. By definition of $I_\vee$:

$$v(F) = \tilde{v}(A \vee B) \overset{\text{def}}{=} I_\vee(\tilde{v}(A), \tilde{v}(B)) \overset{\text{I.H.}}{=} I_\vee(\tilde{v}'(A), \tilde{v}'(B)) \overset{\text{def}}{=} \tilde{v}'(A \vee B) = v'(F)$$

- If $F = (A \to B)$ for some $A, B \in \mathcal{F}_\mathcal{L}$, then each $p \in \mathcal{L}$ occurring in A occurs in F too and each $p \in \mathcal{L}$ occurring in B occurs in F too. So we can apply the I.H. $\tilde{v}(A) = \tilde{v}'(A)$ and $\tilde{v}(B) = \tilde{v}'(B)$. By definition of $I_\to$:

$$v(F) = \tilde{v}(A \to B) \overset{\text{def}}{=} I_\to(\tilde{v}(A), \tilde{v}(B)) \overset{\text{I.H.}}{=} I_\to(\tilde{v}'(A), \tilde{v}'(B)) \overset{\text{def}}{=} \tilde{v}'(A \to B) = v'(F)$$

□

Remarks on the lemma: the lemma allow us **evaluate the behavior of any** $F \in \mathcal{F}_\mathcal{L}$ under all assignments (of which there are infinitely many) **just by computing the truth table of** F (finite object, albeit it can be large), whose columns are indexed by the prepositional letters actually occurring in F.

If the formula F contains exactly the propositional letters $p_1, p_2, \ldots, p_n$, how many rows will the truth table for F have? Based on the principles introduced in 2.4.1, there are 2^n rows. This growth is exponential with respect to the number of propositional letters in F. Each row corresponds to a possible truth value assignment (a function) for $p_1, p_2, \ldots, p_n$ mapping to $\{0,1\}$.

If the formula F contains exactly the propositional letters $p_1, p_2, \ldots, p_n$, how many rows will the truth table for F have? Using what we introduced in 2.4.1, there are 2^n rows. This growth is exponential in the length of F. Each row is a possible assignment of $p_1, \ldots, p_n \mapsto \{0,1\}$.

Example 2.4.9. using Lemma 2.4.3 we can prove using truth table that, let $A, B \in \mathcal{F}_{\mathcal{L}}$:

1. $A \to \neg A$: satisfiable, but not a tautology
2. $A \wedge \neg A$: unsatisfiable
3. $A \to (A \vee B)$: tautology

You can check with truth tables (under each column the result of each operand):

A	A	$\to$	$\neg$	A	
0	0	1	1	0	1
1	1	0	0	1	0

A	A	$\wedge$	$\neg$	A	
0	0	0	1	0	0
1	1	0	0	1	0

A	B	A	$\to$	$(A$	$\vee$	$B)$	
0	0	0	1	0	0	0	1
0	1	0	1	0	1	1	1
1	0	1	1	1	1	0	1
1	1	1	1	1	1	1	1

Below is a series of important tautologies. They are straightforward to prove using truth tables.

- $A \to (B \to A)$ (called "weakening")
- $(\neg A \to A) \to A$ (called "consequentia mirabilis")
- $(A \to B) \vee (B \to A)$ (called "prelinearity")
- $(A \wedge B) \to A$
- $A \to (A \vee B)$
- $A \vee \neg A$ (called "tertium non datur" or "excluded middle")
- $\neg\neg A \to A$ and $A \to \neg\neg A$ (called "double negation laws" or "involutiveness of negation")

Let $A \in \mathcal{F}_{\mathcal{L}}$, $A \to \neg A$, the truth table:

A	A	$\to$	$\neg$	A	
0	0	1	1	0	1
1	1	0	0	1	0

This **does not take into consideration differences between A and simple letters**, but if we instantiate A with a specific formula, for example $A := p \wedge \neg p$ (a contradiction) then: $(p \wedge \neg p) \to \neg(p \wedge \neg p)$ is a tautology. But if $A := p \vee \neg p$ (tauto) then: $(p \vee \neg p) \to \neg(p \vee \neg p)$ is a contradiction.

Fact 2.4.2. A Tautology/Contradiction remains a Tautology/Contradiction if you replace letters with arbitrary formulas (the contrary doesn't necessarily hold).

Two very important Tautologies are:

- **Prelinearity**: $(A \to B) \vee (B \to A) \quad \forall(A, B \in \mathcal{F}_{\mathcal{L}})$. This kinda means that "in every possible world $A \to B$ is true or $B \to A$ is true".
- **Excluded middle**: $A \vee \neg A$. This tautology asserts that for any proposition A, it must be either true or false, there is no middle ground. In logical terms, this means that in every possible world, A is either true or $\neg A$ is true. The rejection of the principle of the excluded middle forms the basis for other logical systems, such as constructive or intuitionistic logic, which are particularly significant in computer science. Unlike classical logic, these systems do not assume $A \vee \neg A$ as universally valid, focusing instead on provability and constructibility.

This page is intentionally left blank.

※ Semantics notions for sets of formulas

Let $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$ (Γ is a subset of the set of all formulas), then Γ is called a **theory**.

Definition 3.0.1. A **theory** $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$ is **satisfiable** iff it exists a truth value assignment $v : \mathcal{L} \to \{0, 1\}$ such that for all $\gamma \in \Gamma$ then $\tilde{v}(\gamma) = 1$ (in symbols $v \models \gamma$ we mean v satisfies γ as a formula) .

Definition 3.0.2. $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$ is **unsatisfiable** iff for all assignments $v : \mathcal{L} \to \{0, 1\}$ there exists at least $\gamma \in \Gamma$ such that $\tilde{v}(\gamma) = 0$.

$v \not\models \Gamma$ for all $v : \mathcal{L} \to \{0, 1\}$. $v \not\models \gamma$ if $\tilde{v}(\gamma) = 0$.

3.1 Logical consequence

Of a formula $F \in \mathcal{F}_{\mathcal{L}}$ from a theory $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$).

Definition: let $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$, $F \in \mathcal{F}_{\mathcal{L}}$, then F **is a logical consequence of** Γ, $\Gamma \models F$, iff for every assignment $v : \mathcal{L} \to \{0, 1\}$ satisfying Γ it holds that v satisfies F too:

$$v \models \Gamma \implies v \models F$$

To express the fact that a given formula $F \in \mathcal{F}_{\mathcal{L}}$ **is a tautology** by means of the notion of logical consequence there are several ways, for example

$$T \models F$$

with T being an always true constant.
Another solution: let Ω be a set of tautologies:

$$\Omega \models F$$

Or even:

$$\emptyset \models F$$

since you can't falsify a formula in the $\emptyset$, there's no formula.
For a **tautology** is also commonly used:

$$\models F$$

with no operand on the left side.

3.1.1 Fundamental semantic properties of theories with respect to logical consequence

Lemma: let $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$ be a theory and $A \in \mathcal{F}_{\mathcal{L}}$, then $\Gamma \models A$ **iff** $\Gamma \cup \{\neg A\}$ is unsatisfiable.

Proof. $\Gamma \models A$ **iff** (by def of $\models$) for all assignments $v : \mathcal{L} \to \{0,1\}$, $\tilde{v}(\gamma) = 1$ for all $\gamma \in \Gamma$ implies $\tilde{v}(A) = 1$.

Iff for all $v : \mathcal{L} \to \{0,1\}$ either it is not the case that $\tilde{v}(\gamma) = 1$ for all $\gamma \in \Gamma$, or $\tilde{v}(A) = 1$.

Iff for all $v : \mathcal{L} \to \{0,1\}$ there exists $\gamma \in \Gamma$ such that $\tilde{v}(\gamma) = 0$ or $\tilde{v}(\neg A) = 0$.

Iff (by definition of unsatisfiability) $\Gamma \cup \{\neg A\}$ is unsatisfiable.

□

Basically, if A is a logical consequence of Γ then there's no case in which $\Gamma \cup \{\neg A\}$ can be true, since for all assignments in which Γ is satisfied A is also satisfied and thus $\neg A$ is always false.

Lemma (Semantical deduction Lemma): Let $P, Q \in \mathcal{F}_{\mathcal{L}}$, $\{P\} \models Q$ iff $P \to Q$. The logical consequence relation can be internalized using the logical implication connective.

Proof. $P \models Q$ **iff** (by definition of $\models$) for all $v : \mathcal{L} \to \{0,1\}$, $\tilde{v}(P) = 1$ implies $\tilde{v}(Q) = 1$.

Iff (by definition of $I_{\to}$) $\tilde{v}(P \to Q) = 1$ for all $v : \mathcal{L} \to \{0,1\}$ that is $P \to Q$ is a tautology.

□

Saying $P \models Q$ or $P \to Q$ is kinda the same.

Theorem (Semantical deduction): Let $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$ and $P, Q \in \mathcal{F}_{\mathcal{L}}$, then $\Gamma \cup \{P\} \models Q$ (which will be also written as $\Gamma, P \models Q$) iff $\Gamma \models P \to Q$.

Proof. $\Gamma, P \models Q$ **iff** (by definition of $\models$) for all assignments $v : \mathcal{L} \to \{0,1\}$ such that $v \models \Gamma$ and $\tilde{v}(P) = 1$ (all assignments that satisfy both Γ and P) then $\tilde{v}(Q) = 1$ (also satisfy Q).

Iff (by definition of $I_{\to}$) for all $v : \mathcal{L} \to \{0,1\}$ such that $v \models \Gamma$ it holds that $\tilde{v}(P \to Q) = 1$.

Iff (by definition of $\models$) $\Gamma \models P \to Q$.

□

This is a stronger version of the lemma presented earlier.

3.2 Compactness Theorem

It's one of the theorems that lies at the foundation of logic.

Let $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$, Γ **is satisfiable iff** for all $\Gamma' \subseteq \Gamma$, with Γ' finite, then Γ' **is satisfiable**. Every subset of a set of satisfiable formulas is satisfiable.

Do we *really* need to consider the case Γ is infinite (contains infinitely many formulas)? Yeah, kinda, it will be shown that in first order logic a single formula may contain the amount of information of infinitely many propositional formulas (whatever this means).

Compactness theorem allows us to **reduce the analysis of satisfiability** of a possibly infinite theory **to the analysis of its finite sub-theories**, but observe that **there are infinitely many finite sub-theories of an infinite theory**.

Rewriting the theorem: $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$ **is satisfiable iff** for all $\Gamma' \subseteq_\omega \Gamma$ then Γ' is satisfiable (with $\subseteq_\omega$ meaning finite subset). The non-trivial case is when Γ is infinite.

Proof. Proving the statement **from left to right** is trivial, if $v \models \Gamma$ then, clearly, by the definition of satisfiability and the notion of subset, $v \models \Gamma'$, for each $\Gamma' \subseteq \Gamma$ too (not even $\subseteq_\omega$, just any subset, even infinite).

□

Proving right to left is not so trivial.

Consider the statement **equivalent** to the theorem **obtained via contraposition** ($A \to B$ says the same thing as $\neg B \to \neg A$). Again from left to right: if there is $\Gamma' \subseteq_\omega$ is unsatisfiable then clearly Γ is unsatisfiable.

From right to left: if Γ **is unsatisfiable**, then there **exists** $\Gamma' \subseteq_\omega \Gamma$ such that Γ' is **unsatisfiable**.

Some terminology before the proof: we say that a given theory $\Gamma \subseteq \mathcal{F}_{\mathcal{L}}$ is **finitely satisfiable** (fin sat) **iff** for all finite sub-theories $\Gamma' \subseteq_\omega \Gamma$, Γ' **is satisfiable**.

The proof of the non trivial part consists of Γ fin sat, implies Γ sat.

Proof. We need some facts before the proof.

Preparation: we fix any **arbitrarily chosen listing** $F_1, F_2, \ldots, F_k, \ldots$ (enumerably infinite, without repetitions) of all formulas $\in \mathcal{F}_{\mathcal{L}}$. To concretely implement such an enumeration we have two issues to face:

- 1^{st} issue: the **alphabet of** $\mathcal{F}_{\mathcal{L}}$: $\{\wedge, \vee, \neg, \rightarrow\} \cup \{), (\} \cup \mathcal{L}$. This contains an infinite set an thus is an **infinite alphabet**. To **switch to a finite alphabet**:

  $$\{\wedge, \vee, \neg, \rightarrow\} \cup \{), (\} \cup \{p, l\}$$

  with $p_k \in \mathcal{L}$ written as $p_k = plll \ldots l$, with k many ls (just a unary encoding, we just need some kind of finite encoding, let it be a unary encoding, binary, base 10, hex, ...).

- 2^{nd} issue: take the alphabet $\{a, b, \dots, \}$: list all the words on this alphabet, but they are infinite and we must define an order. If we use a lexicographic ordering we get:

ϵ (empty word)
a
aa
$a...aa$
...
$b \leftarrow$ never reached

So the letter b appears after an infinite amount of words. Using a short-lex ordering might be fairer (each word appears at a finite stage):

ϵ (empty word)
a
...
z
aa
ab
...
zz

First word of length 1, then length 2, ecc.

The construction: we **define** an **infinite succession of theories** (sets of formulas) as follows: the succession

$$D_0, D_1, \ldots, D_k, \ldots$$

Which is inductively defined as follows:

- **Base:**

$$D_0 := \Gamma$$

- **Step:**

$$D_{n+1} := \begin{cases} D_n \cup \{F_{n+1}\} & \text{if } D_n \cup \{F_{n+1}\} \text{ is fin sat} \\ D_n \cup \{\neg F_{n+1}\} & \text{otherwise} \end{cases}$$

And we also define:

$$D := \bigcup_{i \in \mathbb{N}} D_i$$

Note that this definition is well given and does not directly imply that $D_n \cup \{\neg F_{n+1}\}$ is fin sat, but just that $D_n \cup \{F_{n+1}\}$ is not fin sat.

Now we shall prove some facts about the sequence

$$\Gamma = D_0 \subseteq D_1 \subseteq D_2 \subseteq \ldots \subseteq D_k \subseteq \ldots \subseteq D$$

To prove Γ fin sat implies Γ sat, we first need to prove some facts about the sequence above.

Fact 1: every member of the sequence D_n (for all $n \in \mathbb{N}$) is himself **fin sat**.

Proof. By induction on n

- **Base:** $n = 0$, D_0 is fin sat, by definition, since $D_0 = \Gamma$ and Γ is fin sat.
- **Step:** assume that Fact 1 is true for $D_0, D_1, \ldots, D_n$ and prove it for D_{n+1}. To prove this we shall proceed by contradiction. So we assume D_{n+1} is not fin sat with the aim of finding a contradiction.

  Absurdum Hypothesis: D_{n+1} is not fin sat.

  Hence by the definition of the sequence, $D_n \cup \{F_{n+1}\}$ and $D_n \cup \{\neg F_{n+1}\}$ are both NOT fin sat.

  Now we use the definition of fin sat, they are not fin sat
  iff there exists
  $$D' \subseteq_\omega D_n \cup \{F_{n+1}\}$$
  such that D' is unsatisfiable.

  Iff there exists
  $$D'' \subseteq_\omega D_n \cup \{\neg F_{n+1}\}$$
  such that D'' is unsatisfiable.

  "Lazy" version, we can assume, without loss of generality that $F_{n+1} \in D'$, $\neg F_{n+1} \in D''$. We can argue that they are already in there, otherwise we can add them without major problems (if it's unsatisfiable adding something keeps it that way).
  The actual version: if $F_{n+1} \notin D'$ then $D' \subseteq D_n$, but D_n is fin sat, so D' is fin sat, which is a contradiction.

  That is we can display D' and D'' as follows
  $$D' = E' \cup \{F_{n+1}\}, \quad D'' = E'' \cup \{\neg F_{n+1}\}$$
  with $E' = D' \setminus \{F_{n+1}\}$ and $E'' = D'' \setminus \{\neg F_{n+1}\}$.

  We observe that by construction that $E', E'' \subseteq_\omega D_n$ they are subsets of D_n.

Note: if we take the **theory** $E' \cup E'' \cup \{F_{n+1}\}$ this is **unsatisfiable**, since it contains D'. For the same reason $E' \cup E'' \cup \{F_{n+1}\}$ is **also unsatisfiable** since it contains D''.

But $E' \cup E''$ is a finite subset of D_n, $\subseteq_\omega D_n$, so they **must be fin sat** since we know that D_n is fin sat by I.H. $E' \cup E''$ **is satisfiable**.

This **satisfiable** set **differs** from the **unsatisfiable** ones by **one formula**, notably the same formula.

If we find an assignment that satisfies F_{n+1} then D' is satisfiable, otherwise $\neg F_{n+1}$ is satisfied and D'' becomes satisfiable.

So, we have:
- $E' \cup E''$ **satisfiable**
- $E' \cup E'' \cup \{F_{n+1}\}$ **unsatisfiable** (†)
- $E' \cup E'' \cup \{\neg F_{n+1}\}$ **unsatisfiable** (††)

And there exists (by definition of satisfiability) $v : \mathcal{L} \to \{0, 1\}$ such that $v \models E' \cup E''$ ($\tilde{v}(F) = 1$ for all $F \in E' \cup E''$). We can have a **truth assignment** that **satisfies** all formulas in that **set**.

What about $\tilde{v}(F_{n+1})$?
- if it's 0 it will satisfy $\tilde{v}(\neg F_{n+1})$ and then $v \models E' \cup E'' \cup \{\neg F_{n+1}\}$, **contradicting** (††)
- If it's 1 it will satisfy $\tilde{v}(F_{n+1})$ and then $v \models E' \cup E'' \cup \{F_{n+1}\}$, **contradicting** (†)

And, by **bivalence**, there are **no other cases to consider**. We have reached in any case the contradiction that proves the induction step (D_{n+1} is actually fin sat).
This concludes the proof by induction of Fact 1.

We have proved that **each set** in our construction D_i **is fin sat**.

□

Fact 2: remembering the definition of D, D is fin sat. We proved that **each set is fin sat**, now we need to **prove** that **their union is fin sat**

$$D = \bigcup_{i \in \mathbb{N}} D_i \quad \text{id fin sat}$$

Do we *really* need to prove this? Yes, there are some properties of some sets, such as finiteness, that don't live into their union (with a collection of finite sets, their union might not be a finite set). **Properties of the members don't automatically carry over to their union**.

Proof. Let D' be a finite subset of D ($D' \subseteq_\omega D$). It is sufficient to prove that D' is satisfiable.

We can display the finite set D' by listing its formulas:

$$D' = \{F_{i1}, F_{i2}, \ldots, F_{im}\} \text{ for some } m \in \mathbb{N}$$

We give these indexes since we don't know exactly what formulas are inside D', but there must be a maximum, whatever that number is and whatever the numbers in the set are.

Let k be the maximum index

$$k = \max_{j=1}^{m}(ij)$$

Then

$$D' \subseteq \{F_1, F_2, \ldots, F_k\} \cap D \subseteq D_k$$

The set D' must be a subset of all the formulas between 1 and its highest index k, which is a subset of D_k, by definition f D_k.

So $D' \subseteq_\omega D_k$ (as D_k is defined in our sequence), but by Fact 1 D_k is fin sat. So D' is satisfiable since it's one of its sub-theories.

□

Fact 3: For each formula $F_t \in \mathcal{F}_\mathcal{L}$, **exactly one among** F_t and $\neg F_t$ **belongs to** D.

Proof. By construction of the sequence, $F_t \in D_t$ or $\neg F_t \in D_t$. Since $D_t \subseteq D$: $F_t \in D$ or $\neg F_t \in D$. This proves that at least one is in the set.

We now have to exclude that both of them are in the set: ($\{F_t, \neg F_t\} \subseteq D$).
But D is satisfiable, and clearly $\{F_t, \neg F_t\}$ would contradict Fact 2 since it can't be satisfiable, by the definition of $I_\neg$.

So $F_t \in D$ or $\neg F_t \in D$ and $\{F_t, \neg F_t\} \notin D$. Exactly one is $\in D$.

□

Construction: we define the following **assignment** $v_D : \mathcal{L} \to \{0, 1\}$.
For all $p \in \mathcal{L}$, $v_D(p) = 1$ iff $p \in D$.

Notice that v_D is **well defined** (either $p \in D$ or $p \notin D$ for each possible $p \in \mathcal{L}$).

Fact 4: $v_D \models D$ (for all $F \in D$, $\tilde{v}_D(F) = 1$); it's an assignment that satisfies D. If we can **prove this** then we have **proved that** $v_D \models \Gamma$ **since** $\Gamma \subseteq D$ (by construction of D). So Γ is sat.

Proof. By structural induction, we prove that for all formulas $F \in \mathcal{F}_\mathcal{L}$: $\tilde{v}_D(F) = 1$ **iff** $F \in D$.

Notice: it would suffice to prove just one side of the iff, namely the side that says if $F \in D$ then $\tilde{v}_D(F) = 1$ (but the stronger proof is easier). We ask for the biconditional (iff) for better using the I.H.

Base: $F = p$, with $p \in \mathcal{L}$: so the statement we need to prove is the very definition of $\tilde{v}_D(p)$:

$$\tilde{v}_D(p) = v_D(p) = 1 \text{ iff } p \in D$$

Step(s): we need to consider each way that we can decompose a formula and prove that it's $\in D$:

- $F = (\neg G)$, F is the negation of some formula G.
  $\tilde{v}_D(F) = 1$ **iff** (by definition of $I_\neg$) $\tilde{v}_D(G) = 0$ **iff** (by I.H.) $G \notin D$ **iff** (by Fact 3) $\neg G \in D$ **iff** $F \in D$ (since $\neg G$ is F).

- $F = (G \wedge H)$.
  $\tilde{v}_D(F) = 1$ **iff** $\tilde{v}_D(G \wedge H) = 1$ **iff** (by definition of $I_\wedge$) $\tilde{v}_D(G) = 1$ and $\tilde{v}_D(H) = 1$ **iff** (by I.H.) $G \in D$ and $H \in D$ **iff** $G \wedge H \in D$ **iff** $F \in D$.
  To say that $G \wedge H \in D$ we can observe (by contradiction) that if $G \wedge H \notin D$ **iff** (by Fact 3) $\neg(G \wedge H) \in D$, then $\{G, H, \neg(G \wedge H)\} \subseteq_\omega D$ and this is a contradiction since by Fact 2 D is fin sat, and this subset is unsatisfiable.
  Then if $G \wedge H \in D$ then $G, H \in D$ (otherwise $G \notin D$ or $H \notin D$). Assume $G \notin D$ (the case $H \notin D$ is analogous), by Fact 3, $\neg G \in D$ and then $\{G \wedge H, \neg G\} \subseteq_\omega D$, which is

unsatisfiable, contradicting Fact 2 again. So $F \in D$.

- $F = (G \vee H)$.
  $\tilde{v}_D(G \vee H) = 1$ iff (by definition of $I_\vee$) $\tilde{v}_D(G) = 1$ or $\tilde{v}_D(H) = 1$ **iff** (by I.H.) $G \in D$ or $H \in D$ **iff** $G \vee H \in D$ **iff** $F \in D$.
  If $G \in D$ or $H \in D$ then $(G \vee H) \in D$. By contradiction $(G \vee H) \notin D$, then, by Fact 3, $\neg(G \vee H) \in D$, then
  - if $G \in D$ then $\{G, \neg(G \vee H)\} \subseteq_\omega D$, which contradicts Fact 2
  - if $H \in D$ then $\{H, \neg(G \vee H)\} \subseteq_\omega D$, which contradicts Fact 2

  If $(G \vee H) \in D$ then $G \in D$ or $H \in D$. By contradiction $G \notin D$ and $H \notin D$ and thus $\{\neg G, \neg H, (G \vee H)\} \subseteq_\omega D$, which is unsatisfiable and contradicts Fact 2.

- Prove implication for fun (remember that $G \to H$ has the same truth table as $(\neg G) \vee H$)

□

We have proved Fact 4: $v_D \models D$.

Then a fortiori (for stronger reasons) $v_D \models \Gamma$ (the assignment satisfies D, which is larger that Γ, so v_D must satisfy Γ).

So we have proved that:

$$\Gamma \text{ fin sat} \Leftrightarrow \Gamma \text{ sat}$$

□

By **contraposition**:

$$\Gamma \text{ unsat} \implies \text{there exists } \Gamma' \subseteq_\omega \Gamma, \ \Gamma' \text{ unsat}$$

Remarks on Compactness Theorem:

$$\Gamma \subseteq \mathcal{F}_\mathcal{L} \qquad \Gamma' \subseteq \Gamma$$

We **proved the theorem** not by looking inside Γ, but by **enlarging it** with our sequence of D_is, and **proved that** $v_D \models D$, and so $v_D \models \Gamma$, since $\Gamma \subseteq D$.

D is "large" and satisfiable. We **cannot expand** D **further without losing** the property of D **being satisfiable**.

Proof. First of all, is D everything? Does it contain it all? No, it doesn't, look at Fact 3, so $D \neq \mathcal{F}_\mathcal{L}$.

Let $F_t \notin D$. By Fact 3: $\overline{D} = D \cup \{F_t\}$, and thus $\neg F_t \in D$.
Then $\{F_t, \neg F_t\} \subseteq_\omega \overline{D}$ so $\overline{D}$ is unsatisfiable.
If I add anything not already in D, the negation of that formula must already be in D so the set cannot remain satisfiable if I add anything else.

□

D is a **maximal expansion**, with respect to **satisfiability**, of Γ.

We call a set E of formulas **maximally satisfiable** if it has the same property of D: **iff** E is a set and for all $F \in \mathcal{F}_{\mathcal{L}}$, $F \notin E$ the set $E \cup \{F\}$ is unsatisfiable (if I add anything the set cannot remain satisfiable).

D is **a** maximal expansion of Γ, there can be **many**, in our proof D is fixed, after fixing the listing of formulas.

Let us see that there are **different maximal expansions** of Γ. For instance $\Gamma = \{p_{27}\}$, we have 2 listings:

- listing 1: p_{13}, p_{14}
- listing 2: $\neg p_{13}$, $\neg p_{14}$

Then for listing 1: D_1 is expanded with p_{13} while the D_1 made from listing 2 is expanded with $\neg p_{13}$ so, in the final set D, for listing 1 $p_{13} \in D$ while for listing 2 $\neg p_{13} \in D$. We can expand our set in different ways, based on which listing I choose.

If we reason about the **information contained** inside a **maximally satisfiable theory** D we notice that it carries the **same amount of information** as a **truth value assignment** $v : \mathcal{L} \to \{0, 1\}$.

D max sat theory, from this we defined v_D as $v_D(p) = 1$ iff $p \in D$ for all $p \in \mathcal{L}$. We can go the other way around, **starting from any assignment** $v : \mathcal{L} \to \{0, 1\}$ and **construct the set of formulas** D_v:

$$D_v := \{F \in \mathcal{F}_{\mathcal{L}} : \tilde{v}(F) = 1\}$$

This way

- D_v is a max sat theory
- $v_{D_w} = w$, with w truth assignment $w :\to \{0, 1\}$
- same thing with: $D_{v_E} = E$ with E a max sat theory

Associated with every max sat theory there is an assignment satisfying the theory ($v_D \models D$), and similarly for every assignment there can be a max sat theory satisfied by that assignment.

How can we tell if:

$$(\Gamma \subseteq \mathcal{F}_{\mathcal{L}},\ A \in \mathcal{F}_{\mathcal{L}}) \qquad \Gamma \models A?$$

All formulas in Γ must satisfy also A, so A it's a logical consequence.

Already proven Lemma: $\Gamma \models A$ iff we can **reduce this problem to** the unsatisfiability problem: $\Gamma \cup \{\neg A\}$ **is unsatisfiable**.

Assume Γ is a finite theory (Γ contains finitely many formulas)

$$\Gamma = \{B_1, B_2, \ldots, B_u\}$$

There are finitely many formulas, so there's a last element B_u

$$\Gamma \models A \text{ iff}$$
$$\{B_1, B_2, \ldots, B_u\} \models A \text{ iff}$$
$$\{B_1, B_2, \ldots, B_u, \neg A\} \text{ is unsat, as a theory, iff}$$

by the definition of $\wedge$ ($I_\wedge$)

$$B_1 \wedge B_2 \wedge \ldots \wedge B_u \wedge \neg A \text{ is unsat, as a formula}$$

And this is decidable in finite time, however long it may be.

What if Γ is infinite? The conjunction of infinitely many things it's not a formula and its satisfiability it's not decidable in finite time. So

$$\Gamma \cup \{\neg A\} \text{ is unsat}$$

By compactness theorem, there is

$$\Gamma' \subseteq_\omega \Gamma \cup \{\neg A\} \text{ is unsat}$$

Good news: we can have a finite theory which is unsatisfiable. Bad news: we don't know how to find it since there are infinitely many.
$\rightarrow$ "Semidecidability". If the input is infinite, it's improper speaking of decidability, it's semidecidability. We may be able to tell if it's satisfiable in finite time, or the algorithm could not end.

3.3 Semantic Equivalence (logical equivalence)

Are these the same?

$$A \wedge B \qquad B \wedge A$$

These are syntactically different, they are different strings of symbols.

Semantically, **for all assignments** $v : \mathcal{L} \to \{0, 1\}$ it holds that $\tilde{v}(A \wedge B) = \tilde{v}(B \wedge A)$, by the definition of $I_\wedge$. They always **get the same resulting truth values**.

3.3.1 Equivalence relation

What is a relation? An n-ary relation R on a set S is just a subset of the n-th Cartesian power S^n of S:

$$R \subseteq S^n$$

A relation, mathematically, is just this.

The **equivalence relation**: considering R on a set S, R is a **binary relation** ($R \subseteq S^2$) such that it **satisfies 3 conditions**:

- R is **reflexive**: for all $s \in S$, $(s, s) \in R$
- R is **symmetric**: for all $s_1, s_2 \in S$ if $(s_1, s_2) \in R$ then also $(s_2, s_1) \in R$
- R is **transitive**: for all $s_1, s_2, s_3 \in S$ if $(s_1, s_2) \in R$ and $(s_2, s_3) \in R$ then $(s_1, s_3 \in R)$

Equivalence classes: Let $R \subseteq S^2$ be an equivalence relation, then for each $s \in S$ the **equivalence class** of s, called ($[s]_R$ or ${}^s/_R$) is **defined as**

$$[s]_R := \{t \in S : (s, t) \in R\}$$

Basically, all elements equivalent to s.

Observe: if $(s, t) \in R$ then $[s]_R = [t]_R$ and vice versa, if $[s]_R = [t]_R$ then $(s, t) \in R$.

Partitions of a set: A partition of S is a set $\{B_1, B_2, \ldots, B_k, \ldots\} = \{B_i : i \in I\}$ of subsets of S (not necessarily finite, $B_i \subseteq S$) such that:

- $B_i \cap B_j = \emptyset$ for all $i \neq j \in I$
- $\bigcup_{i \in I} B_i = S$

A partition is a way of **cutting S into blocks**.

Every **equivalence relation $R \subseteq S^2$ determines uniquely a partition** P_R of S

$$P_R = \{[s]_R : s \in S\}$$

The blocks of P_R are the equivalence classes of R.

We can go the other way around, every **partition** P of S **determines uniquely an equivalence relation** R_P on S ($R_P \subseteq S^2$)

$$R_p : (s,t) \in R_P \text{ iff } s,t \text{ belong to the unique same block } B_i \in P$$

3.3.2 Definition of Semantic equivalence

Let us verify the the following **is an equivalence relation on $\mathcal{F}_\mathcal{L}$**

$$\equiv \subseteq \mathcal{F}_\mathcal{L}^2$$

For each $(A,B) \in \mathcal{F}_\mathcal{L}^2$ (or simpler, $A, B \in \mathcal{F}_\mathcal{L}$), then $(A,B) \in \equiv$, also written as $A \equiv B$ (A is **logically equivalent to** B) **iff**, for any $v : \mathcal{L} \to \{0,1\}$, $\tilde{v}(A) = \tilde{v}(B)$.

For all $A \in \mathcal{F}_\mathcal{L}$:

- $\equiv$ is **reflexive**: for all $A \in \mathcal{F}_\mathcal{L}$, $A \equiv A$, for each $v : \mathcal{L} \to \{0,1\}$ $\tilde{v}(A) = \tilde{v}(A)$
- $\equiv$ is **symmetric**: for all $A, B \in \mathcal{F}_\mathcal{L}$, $A \equiv B$ implies $B \equiv A$, for all $v : \mathcal{L} \to \{0,1\}$: $\tilde{v}(A) = \tilde{v}(B)$ implies $\tilde{v}(A) = \tilde{v}(B)$
- $\equiv$ is **transitive**: for all $A, B, C \in \mathcal{F}_\mathcal{L}$, $A \equiv B$ and $B \equiv C$ implies $A \equiv C$, for all $v : \mathcal{L} \to \{0,1\}$: $\tilde{v}(A) = \tilde{v}(B)$ and $\tilde{v}(B) = \tilde{v}(C)$ implies $\tilde{v}(A) = \tilde{v}(C)$

$\equiv$ **determines a partition of** $\mathcal{F}_{\mathcal{L}}$. We subdivided the world in such a way that formulas that say the same thing are in the same block

$$P_{\equiv} = \{[A]_{\equiv} : A \in \mathcal{F}_{\mathcal{L}}\}$$

$$\mathcal{F}_{\mathcal{L}}/_{\equiv} = \{[A]_{\equiv} : A \in \mathcal{F}_{\mathcal{L}}\}$$

$[A \wedge B]_{\equiv} = [B \wedge A]_{\equiv}$ says that the formulas are equivalent, which is the same as saying $A \wedge B \equiv B \wedge A$.

Two formulas are **logically equivalent** if they have the **same truth value under any assignment**.

In general if $R \subseteq S^2$ is an equivalence relation, we write ${}^{S}/_{R} = \{[S]_R : s \in S\}$. This is called "$S$ modulo R" or "the quotient of S over R".

We eventually will enrich the set of logical equivalence classes $\mathcal{F}_{\mathcal{L}}/_{\equiv}$ with operations which shall confer to the enriched set the status of an algebraic structure (namely: Boolean algebra).

3.4 Substitution (letter with a formula)

Substitution of a **propositional letter with a formula** (inside a formula). We want to get new formula from another one, while remaining semantically equivalent.

For each $A \in \mathcal{F}_\mathcal{L}$, for each $p \in \mathcal{L}$, for each $B \in \mathcal{F}_\mathcal{L}$, let us define the formula obtained by **replacing in** A **each occurrence of** p **with** B, $A[{}^{B}\!/_{p}] \in \mathcal{F}_\mathcal{L}$. Define it inductively as follows:

- **Base:** $A = q$, $Q \in \mathcal{L}$, define
$$q[{}^{B}\!/_{p}] := \begin{cases} B & q = p \\ q & q \neq p \end{cases}$$
If $q = p$ then it becomes B, otherwise nothing changes.
- **Step:** cases, if we have to substitute:
$$(\neg A)[{}^{B}\!/_{p}] := \neg(A[{}^{B}\!/_{p}])$$
$$(A_1 \wedge A_2)[{}^{B}\!/_{p}] := A_1[{}^{B}\!/_{p}] \wedge A_2[{}^{B}\!/_{p}]$$
$$(A_1 \vee A_2)[{}^{B}\!/_{p}] := A_1[{}^{B}\!/_{p}] \vee A_2[{}^{B}\!/_{p}]$$
$$(A_1 \to A_2)[{}^{B}\!/_{p}] := A_1[{}^{B}\!/_{p}] \to A_2[{}^{B}\!/_{p}]$$

Lemma (substitution): let $v : \mathcal{L} \to \{0,1\}$ be an assignment and $S, T \in \mathcal{F}_\mathcal{L}$. If $\tilde{v}(S) = \tilde{v}(T)$, then for all $A \in \mathcal{F}_\mathcal{L}$, and all $p \in \mathcal{L}$

$$\tilde{v}(A[{}^{S}/_{p}]) = \tilde{v}(A[{}^{T}/_{p}])$$

Proof. By induction on A:

- **Base:** $A = q$, $q \in \mathcal{L}$, two cases:
  - $q \neq p$, then $q[{}^{S}/_{p}] = q = q[{}^{T}/_{p}]$, then
  $$\tilde{v}(q[{}^{S}/_{p}]) = \tilde{v}(q) = \tilde{v}(q[{}^{T}/_{p}])$$
  - $q = p$, then $q[{}^{S}/_{p}] = S$ and $q[{}^{T}/_{p}] = T$, so
  $$\tilde{v}(q[{}^{S}/_{p}]) = \tilde{v}(S) = \tilde{v}(T) = \tilde{v}(q[{}^{T}/_{p}])$$
- **Step:** cases are:
  - $A = \neg B$, by I.H. $\tilde{v}(B[{}^{S}/_{p}]) = \tilde{v}(B[{}^{T}/_{p}])$ and then
  $$\tilde{v}((\neg B)[{}^{S}/_{p}]) = I_\neg\left(\tilde{v}(B[{}^{S}/_{p}])\right) = I_\neg\left(\tilde{v}(B[{}^{T}/_{p}])\right) = \tilde{v}((\neg B)[{}^{T}/_{p}])$$
  - $A = (B \wedge C)$
  - $A = (B \vee C)$
  - $A = (B \to C)$

  The other cases are left as an exercise for the reader.

□

Theorem (substitution): Let $S, T \in \mathcal{F}_\mathcal{L}$, if $S \equiv T$, then for all $A \in \mathcal{F}_\mathcal{L}$ and all $p \in \mathcal{L}$:

$$A[{}^{S}/_{p}] \equiv A[{}^{T}/_{p}]$$

If in a single formula we **replace a formula with an equivalent** one we get a **distinct** formula **but still equivalent**.

Proof. For all $v : \mathcal{L} \to \{0,1\}$ it holds that $\tilde{v}(s) = \tilde{v}(T)$, since $S \equiv T$. So by the substitution Lemma, for all $v : \mathcal{L} \to \{0,1\}$ we have $\tilde{v}(A[{}^{S}/_{p}]) = \tilde{v}(A[{}^{T}/_{p}])$, and then, by definition of $\equiv$

$$A[{}^{S}/_{p}] \equiv A[{}^{T}/_{p}]$$

□

Theorem: Given $F, G, H \in \mathcal{F}_{\mathcal{L}}$ the following are logical equivalences:

- **Idempotence**: $F \wedge F \equiv F$, $F \vee F \equiv F$
- **Commutativity**: $F \vee G \equiv G \vee F$, $F \wedge G \equiv G \wedge F$
- **Associativity**: $F \wedge (G \wedge H) \equiv (F \wedge G) \wedge H$, $F \vee (G \vee H) \equiv (F \vee G) \vee H$
- **Distributivity**: $F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$, $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
- **Absorption**: $F \vee (F \wedge G) \equiv F$, $F \wedge (F \vee G) \equiv F$

Exercise: prove that idempotence $F \vee F \equiv F$, $F \wedge F \equiv F$ follows only by absorption (using substitution theorem).

- **Double negation** (involutiveness of negation): $\neg\neg F \equiv F$
- **De Morgan**:
$$\begin{cases} \neg(F \vee G) \equiv \neg F \wedge \neg G \\ \neg(F \wedge G) \equiv \neg F \vee \neg G \end{cases}$$
- **Interdefinability**:
$$\begin{cases} F \to G \equiv (\neg F) \vee G \\ F \to G \equiv \neg(F \wedge \neg G) \\ F \vee G \equiv (\neg F) \to G \\ F \wedge G \to \neg(F \to \neg G) \\ F \vee G \equiv \neg(\neg F \wedge \neg G) \\ F \wedge G \equiv \neg(\neg F \vee \neg G) \end{cases}$$

Proof. Just one case (the remaining ones are left as an exercise for the reader). We want to show that

$$\neg(F \vee G) \equiv \neg F \wedge \neg G$$

For all $v : \mathcal{L} \to \{0, 1\}$ $\tilde{v}(\neg(F \vee G)) = 1$

iff by the semantics of $\neg$: $I_{\neg}$: $\tilde{v}(F \vee G) = 0$
iff by the semantics of $\wedge$: $\tilde{v}(F) = 0$ and $\tilde{v}(G) = 0$
iff by the semantics of $\neg$: $\tilde{v}(\neg F) = 1$ and $\tilde{v}(\neg G) = 1$
iff by the semantics of $\wedge$: $\tilde{v}(\neg F \wedge \neg G) = 1$

By bivalence $\neg(F \vee G) \equiv \neg F \vee \neg G$ (we've proved that it holds for 1, the only other possible value is 0 so we don't need to prove that).

□

3.5 Derived connectives

We define these operators as **shortened versions of longer formulas**.

We define $F \leftrightarrow G$, as $(F \to G) \wedge (G \to F)$. Specify

$$I_{\leftrightarrow} : \{0,1\}^2 \to \{0,1\}$$

a	b	$a \leftrightarrow b$
0	0	1
0	1	0
1	0	0
1	1	1

Observation: $\models F \leftrightarrow G$ (is a tautology) **iff** $F \equiv G$ **iff** $\models \neg F \leftrightarrow \neg G$ **iff** $\models G \leftrightarrow F$ **iff** $\models \neg G \leftrightarrow \neg F$.

Observation: $F \models G$ and $G \models F$ **iff** $F \equiv G$.

We define other derived connectives $\bot, \top$. Technically they are zero-ary connectives (and as such they are formulas in themselves), they're **logical constants**, don't need arguments to be complete to a formula.

$$\bot, \top \in \mathcal{F}_{\mathcal{L}}$$

$$I_{\bot} : \{0,1\}^0 \to \{0,1\} \quad \text{identify with 0}$$

$$I_{\top} : \{0,1\}^0 \to \{0,1\} \quad \text{identify with 1}$$

The $\{0,1\}^0$ represents the singleton set, so we can **identify the function with the result**. So

$$I_{\bot} := 0 \qquad I_{\top} := 1$$

The $\top$ is in the **equivalence class of tautologies**:

$$\top \equiv A \quad \text{iff } A \text{ is any tautology } (\models A), \text{ e.g. } \top := p \to p$$

And $\bot$ is in the **equivalence class of contradictions**:

$$\bot \equiv A \quad \text{iff } A \text{ is any contradiction } (\models \neg A), \text{ e.g. } \bot := p \wedge \neg p$$

Some things **about these constants**:

$$\neg\top \equiv \bot, \quad \neg\bot \equiv \top$$

If $\models A$

$$\neg A \equiv A \to \bot$$
$$A \equiv \top \to A$$

Some other equivalences:

- **Complementation**

$$A \wedge \neg A \equiv \bot$$
$$A \wedge \neg A \equiv \top$$

- **Unit** and **neutral elements**

$$A \wedge \bot \equiv \bot$$
$$A \vee \bot \equiv A$$
$$A \wedge \top \equiv A$$
$$A \vee \top \equiv \top$$

3.6 $\equiv$ as a congruence

$\equiv$ is **more than** simply being an **equivalence relation**.

$\equiv$ is a **congruence** with respect to $\vee, \wedge, \rightarrow, \neg$ **thought as operations**, so if the operands are thought as

$$\wedge : \mathcal{F}_{\mathcal{L}}^2/\equiv \rightarrow {}^{\mathcal{F}_{\mathcal{L}}}/_{\equiv}$$
$$([A]_{\equiv}, [B]_{\equiv}) \rightarrow [A \wedge B]_{\equiv}$$

$\equiv$ is a congruence with respect to $\wedge$ as defined above.
If $A \equiv B$ and $C \equiv D$ then $A \wedge B \equiv C \wedge D$.

Exercise: verify that $\equiv$ is a congruence with respect to all the connectives $\wedge, \vee, \rightarrow, \neg$ (also $\top, \bot, \leftrightarrow$ even if they're derived so there would technically be no need).

Proof. We demonstrate just the case of $\equiv$ being a congruence with respect to $\neg$.
$A \equiv B$ **iff** $\neg A \equiv \neg B$. For all $v : \mathcal{L} \rightarrow \{0, 1\}$, $\tilde{v}(A) = \tilde{v}(B)$ **iff** $T_{\neg}(\tilde{v}(A)) = I_{\neg}(\tilde{v}(B))$ **iff** $\tilde{v}(\neg A) = \tilde{v}(\neg B)$.

□

Note: it is not true that every equivalence relation is a congruence (with respect to some operations).

We've started by considering the set of particular strings $\mathcal{F}_{\mathcal{L}}$, now we have a set of classes of logical equivalence ${}^{\mathcal{F}_{\mathcal{L}}}/_{\equiv}$ and the operator $\equiv$ can be thought as a congruence.

3.7 Partial order relation

Let R be a **binary relation** on S, $R \subseteq S^2$ such that

- R is reflexive $(s, s) \in R$ for all $s \in S$
- R is **antisymmetric** for all $(s, t) \in R$, if $(s, t) \in R$ and $(t, s) \in R$ then $s = t$
- R is transitive $(s, t) \in R$ and $(t, r) \in R$ then $(s, r) \in R$

Any **relation with these properties is a partial order relation**. The difference between this and an equivalence relation is the antisymmetric property, which states that if two elements are in the relation in a certain order, the opposite ordering can't be in the relation as well.

Symbols: $\leq, \preceq, \sqsubseteq$.

Some examples of partial orderings for the naturals:

$$(\mathbb{N}, \leq)$$

$a \leq b$ if it exists $c \in \mathbb{N}$ such that $a + b = c$.
This relation has a minimum element, which is 0.

$$(\mathbb{N}, \geq)$$

$a \geq b$ if it exists $c \in \mathbb{N}$ such that $a - b = c$.
In this relation 0 is the maximum element.

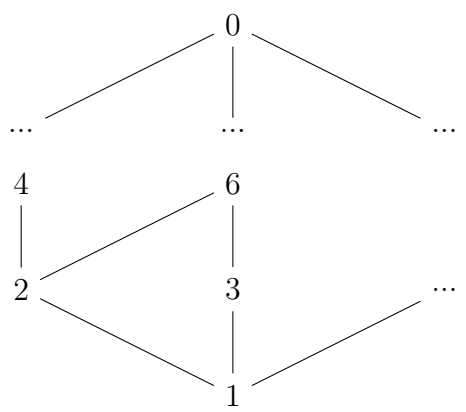
$$(\mathbb{N}, =)$$

In this relation every element is on the same level.

Divisibility order:

$$(\mathbb{N}, |)$$

$a|b$ (a divides b) if there is $c \in \mathbb{N}$ such that $a \cdot c = b$.



This order has 1 as a minimum, all the prime numbers in the first level (atoms), then all numbers created just by primes and so on until the maximal element, which is zero since there's always a number such that $n \cdot 0 = 0$.

A **set** P, **equipped with a partial order relation** $\leq \subseteq P^2$ is called a **partially ordered set**, for short **poset** $(P, \leq)$.

3.8 Lattice

To be a **lattice** there must be, **for each element in the poset**, a **greatest lower bound** and a **lowest upper bound**.

Considering $S \subseteq P$, P poset, $(P, \leq)$.

We can define the **infimum** (greatest lower bound): $\inf S$ if it exists is the element $t \in P$ such that for all $z \in P$, if $z \leq s$ for all $s \in S$ then $z \leq t$ (t is the greatest lower bound).

We can also define the **supremum** $\sup S$: if it exists is the element $t \in P$ such that for all $z \in P$, if $s \leq z$ for all $s \in S$ then $t \leq z$ (t is the lowest upper bound).

$$\inf S = \max\{z \in P : z \leq s \text{ for all } s \in S\}$$
$$\sup S = \min\{z \in P : s \leq z; \text{ for all } s \in S\}$$

If they exist.

We can also define:

$$\min S = t \text{ iff } t = \inf S \text{ and } t \in S$$
$$\max S = t \text{ if } t = \sup S \text{ and } t \in S$$

Example of a set lacking sup (and then lacking max)

$$S \subseteq \mathbb{Q} \qquad S := \{x : x^2 \leq 2\}$$

This has a sup, but not in the rationals. It would be $\sqrt{2} \in \mathbb{R} \setminus \mathbb{Q}$.

While

$$S' \subseteq \mathbb{Q} \qquad S' := \{x : x \lneq 1\}$$

Has a sup but not a max, $\sup S' = 1$ but $1 \notin S'$, so $\max S'$ does not exist.

$$S'' \subseteq \mathbb{Q} \qquad S'' := \{x : x \leq 1\}$$

$\sup S'' = 1$, $1 \in S''$ so $\max S'' = 1$

Definition of Lattice (as poset): A poset $(R, \leq)$ **is a lattice iff** for all $x, y \in R$ there **exists both** $\inf\{x, y\}$ **and** $\sup\{x, y\}$.

Exercise: is our poset a lattice?

$$(\mathbb{N}, \leq)$$

is a lattice because every pair has an greatest lower bound (the smaller number) and a lowest upper bound (the largest number).

$$(\mathbb{N}, |)$$

Divisibility is a lattice, the inf is the greatest common divisor, while the sup is the lowest common multiple.

$$(\mathbb{N}, =)$$

Is NOT a lattice, every element is on the same level, there is no inf or sup.

3.9 Algebraic Structure

(Very sketchy) Definition of Algebraic Structure: Let S be a set, and $*_1, *_2, \ldots, *_n$ a number of operations (can be infinite) on S (that is functions $*_i : S^{\nu(i)} \to S$, where $\nu(i)$ gives the arity of the operation).

Then $(S, *_1, *_2, \ldots, *_n)$ is an **algebraic structure**, or is an algebra.

By algebra we basically mean a set of operations.

Example:

$$(\mathbb{Z}, +, -, 0)$$

$$+ : \mathbb{Z}^2 \to \mathbb{Z} \quad (a, b) \to a + b$$

$$- : \mathbb{Z}^1 \to \mathbb{Z} \quad a \to -a$$

$$0 : \mathbb{Z}^0 \to \mathbb{Z} \quad 0 \text{ the constant/integer number } 0$$

The arity of these operations is, respectively: $\nu(+) = 2$, $\nu(-) = 1$, $\nu(0) = 0$.

3.9.1 Isomorphism

Two **structures** are **isomorphic**

$$(S, *_1, *_2, \ldots, *_n) \cong (R, \circledcirc_1, \circledcirc_2, \ldots, \circledcirc_m)$$

If they mathematically behave in the same manner, i.e. **iff**

$$m = n$$

$$\nu_S^{(i)} = \nu_R^{(i)} \quad \text{for each } i = 1, 2, \ldots, n$$

$$\text{there exists } f : S \to R \text{ such as:}$$

$$\begin{cases} f: \text{ injective:} & x \neq y \in S \implies f(x) \neq f(y) \\ f: \text{ surjective:} & \text{for all } y \in R \text{ there is } x \in S \text{ s.th. } f(x) = y \end{cases}$$

$$\implies f \text{ is bijective (1-1 correspondance)}$$

$$f: \text{ is a homomorphism} : f(*_i(s_1, \ldots, s_k)) = \circledcirc_i(f(s_1), f(s_2), \ldots, f(s_k))$$

$$\nu_S(i) = k, \text{ for all } (s_1, \ldots s_k) \in S^k$$

TLDR: The structures are the same, except for their physical nature, they behave the same as far as those operations are concerned.

3.9.2 Lattices as algebraic structures

A **lattice as an algebraic structure** is a **set with binary operations**

$$(R, \sqcap, \sqcup) \qquad \begin{array}{l} \sqcap : R^2 \to R \\ \sqcup : R^2 \to R \end{array}$$

such that **both** these **operations** are **associative and commutative**.

$\sqcap, \sqcup$ are related by absorption:

$$\text{for all } \; x, y \in R \qquad \begin{cases} x \sqcap (x \sqcup y) = x \\ x \sqcup (x \sqcap y) = x \end{cases}$$

A **lattice as a poset** can be **redefined as/associated** to a **lattice as an algebraic structure**

$$(R, \leq) \leftrightsquigarrow (R, \sqcap, \sqcup)$$

each lattice meant as a poset can be thought of uniquely as an algebra and vice versa.

To do

$$(R, \leq) \rightsquigarrow (R, \sqcap, \sqcup)$$

we just have to define the operations. For all $a, b \in R$ we need to define:

$$a \sqcap b := \inf\{a, b\}$$

$$a \sqcup b := \sup\{a, b\}$$

The rest is an exercise for the reader.

While to do

$$(R, \sqcap, \sqcup) \rightsquigarrow (R, \leq)$$

For all $a, b \in R$, $a \leq b$ iff $a = a \sqcap b$ (or, equivalently, $b = a \sqcup b$).

Check: $a \leq b$ defined as above is such that $\leq$ is

- reflexive: for all $a \in R$, $a \leq a$ iff $a = a \sqcap a$ (by idempotence)
- antisymmetric: ...
- transitive: ...

Doing this we prove that this thing is a poset, but the lattice is a special kind of poset. So now we need to check that the poset $(R, \leq)$ is indeed a lattice. We need to prove that for each pair of element there is an inf and sup. **For each** $a, b \in R$ **there is** $\inf\{a, b\}$ **and** $\sup\{a, b\}$.

Letting $c \leq a$ and $c \leq b$ we have to show that $c \leq a \sqcap b$, since it means we've proven that $a \sqcap b$ is $\inf\{a, b\}$, so it exists and is defined, $\inf\{a, b\} = a \sqcap b$.

$c = a \sqcap c$, by definition of $\leq$, and from the fact that $c \leq b$ and thus $c = b \sqcap c$ so

$$c = a \sqcap c = a \sqcap (b \sqcap c) = (a \sqcap b) \sqcap c \text{ iff } c \leq a \sqcap b$$

So inf exists and is defined.

The proof that $\sup\{a, b\} = a \sqcup b$ is analogous.

We have proved that $(R, \leq)$ is a lattice (as a poset).

3.10 Boolean Algebras

It's an algebra, so a set with some operations and two constants

$$(S, \sqcap, \sqcup, ()^c, 0, 1)$$

The arity of the operations is:

- $\sqcap, \sqcup : S^2 \to S$
- $()^c : S \to S$
- $0, 1 : S^0 \to S$

(The $()^c$ is the complement).

Additional properties: $(S, \sqcap, \sqcup)$ is a lattice (has an order), which is

- **Bounded:** for all $a \in S$ there's a minimum $0 \leq a$ and a maximum $a \leq 1$ such that

$$\begin{cases} a \sqcap 0 = 0 & a \sqcup 1 = 1 \\ a \sqcup 0 = a & a \sqcap 1 = a \end{cases}$$

- **Complemented:** for all $a \in S$, such that $a \sqcap a^c = 0$ and $a \sqcup a^c = 1$.

- **Distributive:** for all $a, b, c \in S$, $a \sqcap (b \sqcup c)$

$$a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$$
$$a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$$

So, boolean algebra is a shorthand name for bounded, complemented and distributive lattices.

Examples of boolean algebra: Let A be a set

$$(\mathcal{P}(A), \cap, \cup, \overline{()}, \emptyset, A)$$

Let $\mathcal{P}(A)$ be the powerset of A (set of all subsets of A).
Check that this is a boolean algebra. It must be commutative, absorption, bounds, ...

If A is finite, say $|A| = n$, then $|\mathcal{P}(A)| = 2^n$. 2^A is a commonly used notation for the powerset.

Every finite Boolean algebra has 2^k elements for some $k \in \mathbb{N}$.

Every finite boolean algebra can be conceived/is **isomorphic with** the **powerset of some finite set**.

$(\mathcal{P}(A), \subseteq)$, S, T, $S \subseteq T$ iff $S = S \cap T$ (or $T = S \cup T$). This is the lattice isomorphic to the boolean algebra defined above.

Example:

$$A = \{a, b\},\ |\mathcal{P}(A)| = |2^A| = 2^2 = 4$$
$$2^A = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$$

So

$$(2^A, \subseteq)$$

is a lattice.

Boolean algebra of truth-values:

$$(\{0,1\}, I_\wedge = \min, I_\vee = \max, I_\neg = 1-, 0 = I_\bot, 1 = I_\top)$$

This is the **smallest significant boolean algebra** and in the ordering we have that 0 is below 1.

It's equivalent to:

$$\cong (\mathcal{P}(\{*\}), \cap, \cup, \overline{()}, \emptyset, \{*\})$$

Boolean algebras of n**-tuples of truth-values**

$$(\{0,1\}^n, \overline{\min}, \overline{\max}, \overline{1-}, (0,0,\ldots.,0), (1,1,\ldots,1))$$

Operations are defined pointwise (componentwise):

$$\overline{\min}((b_1,\ldots,b_n),(c_1,\ldots c_n)) := (\min\{b_1,c_1\},\ldots,\min\{b_n,c_n\})$$

$$\overline{\max}((b_1,\ldots,b_n),(c_1,\ldots c_n)) := (\max\{b_1,c_1\},\ldots,\max\{b_n,c_n\})$$

$$\overline{1-(b_1,\ldots,b_n)} = (1-b_1,\ldots,1-b_n)$$

3.10.1 Lindembaum Algebras of (equivalence classes of) formulas

It is a boolean algebra.

$$(\mathcal{F}_{\mathcal{L}}/\equiv, \wedge, \vee, \neg, [\bot]_{\equiv}, [\top]_{\equiv})$$

Since $\equiv$ is a congruence with respect to $\wedge, \vee, \neg$ the following **definition is well given** (does not depend on the representatives of the classes).

For all $A, B \in \mathcal{F}_{\mathcal{L}}$:

$$[A]_{\equiv} \wedge [B]_{\equiv} := [A \wedge B]_{\equiv}$$
$$[A]_{\equiv} \vee [B]_{\equiv} := [A \vee B]_{\equiv}$$
$$\neg[A]_{\equiv} := [\neg A]_{\equiv}$$

It has an **order relation** since it's also a **lattice**:

$$(\mathcal{F}_{\mathcal{L}}/\equiv, \leq) \quad [A]_{\equiv} \leq [B]_{\equiv} \text{ iff } [A]_{\equiv} = [A \wedge B]_{\equiv} \text{ or } [B]_{\equiv} = [A \vee B]_{\equiv}$$

Some combinatorics about $\mathcal{F}_{\mathcal{L}}/\equiv$

$$|\mathcal{F}_{\mathcal{L}}| = |\mathcal{L}| = |\mathbb{N}| = \aleph_0$$

$$|\mathcal{F}_{\mathcal{L}}/\equiv| = \aleph_0$$

No powerset has cardinality $\aleph_0$, so $|\mathcal{F}_{\mathcal{L}}/\equiv|$ is **not isomorphic** with any powerset algebra.

By $\mathcal{F}_{\mathcal{L}}^{(n)}$ we mean the set of all formulas built using only the first n propositional letters $\{p_1, p_2, \ldots, p_n\}$. Check that

$$|FL^{(n)}/\equiv|$$

is a boolean algebra (called Lindenbaum algebra over the first n letters).

And he checks that, but I'm not paid enough to write it.

First exercise: taking

$$(\mathcal{F}_{\mathcal{L}}/\equiv, \leq)$$

thought as a poset, we know that

$$[A]_{\equiv} \leq [B]_{\equiv}$$

iff $[A]_{\equiv} = [A \wedge B]_{\equiv}$
iff $A \equiv A \wedge B$
iff $\models A \leftrightarrow (A \wedge B)$
iff $\models A \rightarrow B$

Looking at the truth table, we can see that it's structured in such a way that no assignment can make $A \rightarrow B$ a tautology.

Exercise: $F^{(n)}/\equiv$ is isomorphic with

$$2^{2^n} \cong (\mathcal{P}(\mathcal{P}(\,...\,)), \cap, \cup, ()^c, \emptyset, \,...)$$

Some other notable logically equivalent formulas:

- **De Morgan Laws on n many variables**:

$$\neg(F_1 \vee F_2 \vee \,...\,, \vee F_n) \equiv \neg F_1 \wedge \neg F_2 \wedge \,...\, \wedge \neg F_n$$

$$\neg(F_1 \wedge F_2 \wedge \,...\,, \wedge F_n) \equiv \neg F_1 \vee \neg F_2 \vee \,...\, \vee \neg F_n$$

- **Generalized distributivity**:

$$(F_1 \vee \,...\, \vee F_u) \wedge (G_1 \vee \,...\, \vee G_v) \equiv \bigvee_{i=1}^{u} \bigvee_{j=1}^{v} (F_i \wedge G_j)$$

$$(F_1 \wedge \,...\, \wedge F_u) \vee (G_1 \wedge \,...\, \wedge G_v) \equiv \bigwedge_{i=1}^{u} \bigwedge_{j=1}^{v} (F_i \vee G_j)$$

3.11 Term Function

Motivations: $\models F$ **iff** (by definition) for all $v : \mathcal{L} \to \{0,1\}$ it holds that $\tilde{v}(F) = 1$.

We want to look at the **assignment as a function**, and F **as the argument**.
We desire that

- the **formula** "becomes" the **function**
- the **assignments** "become" the **arguments**

We fix a **formula** $A \in \mathcal{F}_{\mathcal{L}}$ and let $Var(A)$ be the set of **propositional letters actually occurring** in A (since assignments are infinite by definition). Now fix a **suitable natural number** $n \in \mathbb{N}$ such that $Var(A) \subseteq \{p_1, p_2, \ldots, p_n\}$ (just take n "large enough" for A).

Desideratum (what we want): we want to **define a function** $\hat{A} : \{0,1\}^n \to \{0,1\}$ **such that** for all assignments $v : \mathcal{L} \to \{0,1\}$:

$$\hat{A}(v(p_1), v(p_2), \ldots, v(p_n)) = \tilde{v}(A)$$

The **formula** has **become a function**, the **assignment** has become the **argument**. Sometimes it's more convenient.

Exercise: verify that assignments, restricted to the first n propositional letters (that is $v : \{p_1, \ldots. p_n\} \to \{0,1\}$), are in bijection with points in $\{0,1\}^n$ (n-tuples of bits).

Definition: of **term function** $\hat{A}$ associated with the formula $A \in \mathcal{F}_{\mathcal{L}}$ with respect to $n \in \mathbb{N}$ ($Var(A) \subseteq \{p_1, \dots, p_n\}$).

We define this by **induction on the structure of** A:

- **Base**: $A = p_i$, $p_i \in \{p_1, \dots, p_n\}$

  $$\hat{A} : \{0,1\}^n \to \{0,1\}$$

  $$\hat{p_i} : \{0,1\}^n \to \{0,1\}$$

  for all $(t_1, \dots, t_n) \in \{0,1\}^n$

  $$\hat{p_i}(t_1, \dots, t_n) := t_i$$

  so it satisfies the desideratum:

  $$\hat{p_i}(v(p_1), \dots, v(p_i), \dots, v(p_n)) = v(p_i) = \tilde{v}(p_i)$$

- **Steps**: we have to define the cases:
  - $A = \neg B$

    $$\hat{A}(t_1, \dots, t_n) = (\hat{\neg B})(t_1, \dots, t_n) := I_{\neg}(\hat{B}(t_1, \dots, t_n)) = 1 - (\hat{B}(t_1, \dots, t_n))$$

  - $A = B \wedge C$

    $$\hat{A}(t_1, \dots, t_n) = (B \hat{\wedge} C)(t_1, \dots, t_n) := I_{\wedge}(\hat{B}(t_1, \dots, t_n), \hat{C}(t_1, \dots, t_n))$$

  - $A = B \vee C$

    $$\hat{A}(t_1, \dots, t_n) = (B \hat{\vee} C)(t_1, \dots, t_n) := I_{\vee}(\hat{B}(t_1, \dots, t_n), \hat{C}(t_1, \dots, t_n))$$

  - $A = B \to C$

    $$\hat{A}(t_1, \dots, t_n) = (B \hat{\to} C)(t_1, \dots, t_n) := I_{\to}(\hat{B}(t_1, \dots, t_n), \hat{C}(t_1, \dots, t_n))$$

Check that in any case of the steps the desideratum holds (a lot of stuff is written, idk what it means).

3.11.1 Boolean algebra of term functions

$$\hat{\mathcal{F}}_{\mathcal{L}}^{(n)} := \left(\left\{\hat{F} : \{0,1\}^n \to \{0,1\} : F \in \mathcal{F}_{\mathcal{L}}^{(n)}\right\}, \hat{\wedge}, \hat{\vee}, \hat{\neg}, \hat{\top}, \hat{\bot}\right)$$

where

$$(\hat{F} \wedge \hat{G})(t_1, \ldots, t_n) = I_{\wedge}\{\hat{F}(t_1, \ldots, t_n), \hat{G}(t_1, \ldots, t_n)\}$$
$$(\hat{F} \vee \hat{G})(t_1, \ldots, t_n) = I_{\vee}\{\hat{F}(t_1, \ldots, t_n), \hat{G}(t_1, \ldots, t_n)\}$$
$$\hat{\neg}\hat{F}(t_1, \ldots, t_n) = I_{\neg}(\hat{F}(t_1, \ldots, t_n))$$

We can prove that $\hat{\mathcal{F}}_{\mathcal{L}}^{(n)}$ is **isomorphic** with $\mathcal{F}_{\mathcal{L}}^{(n)}/\equiv$ ($\hat{\mathcal{F}}_{\mathcal{L}}^{(n)} \cong \mathcal{F}_{\mathcal{L}}^{(n)}/\equiv$) via: $\hat{F} \mapsto [F]_{\equiv}$ for each $F \in \mathcal{F}_{\mathcal{L}}$.

$$\hat{\mathcal{F}}_{\mathcal{L}}^{(n)} = \left(\{\hat{F} : \{0,1\}^n \to \{0,1\} : F \in FL\}, \hat{\wedge}, \hat{\vee}, \hat{\neg}, \hat{\top}, \hat{\bot}\right)$$

Compare with

$$Func^{(n)} = \left(\{g : \{0,1\}^n \to \{0,1\}\}, \hat{\wedge}, \hat{\vee}, \hat{\neg}, \hat{\top}, \hat{\bot}\right)$$

Check that $Func^{(n)}$ **is a boolean algebra**.

This

$$\mathcal{F}_{\mathcal{L}}^{(n)} \subseteq Func^{(n)}$$

is trivially true since each term function belongs to $Func^{(n)}$, while

$$\mathcal{F}_{\mathcal{L}}^{(n)} \supseteq Func^{(n)}$$

is also true, called **functional completeness of propositional logic**.

So

$$\mathcal{F}_{\mathcal{L}}^{(n)} = Func^{(n)}$$

They're not isomorphic, it's **literally the same algebra**, they have the same universe and same operations, it's not a bijection.

3.12 Functional Completeness Theorem

We have to prove that **for each natural number** $n \in \mathbb{N}$, the set of function $Func^{(n)} \subseteq \hat{\mathcal{F}}_{\mathcal{L}}^{(n)}$.
That is, for each function $g : \{0,1\}^n \to \{0,1\}$ there exists a formula $F \in \mathcal{F}_{\mathcal{L}}^{(n)}$ such that

$$g = \hat{F}$$

(that is, for each n-tuple $(t_1, \ldots, t_n) \in \{0,1\}^n$, $g(t_1, \ldots, t_n) = \hat{F}(t_1, \ldots, t_n)$).

Proof. Let $g : \{0,1\}^n \to \{0,1\}$, let us build $F \in \mathcal{F}_{\mathcal{L}}^{(n)}$ such that $g = \hat{F}$.
We can build truth tables:

$t_1,$	$t_2,$	$\ldots,$	t_n	$g(t_1, \ldots, t_n)$
0	0	$\ldots$	0	b_0
0	0	$\ldots$	1	b_1
		$\vdots$		$\vdots$
a_{i1}	a_{i2}	$\ldots$	a_{in}	b_i
		$\vdots$		$\vdots$
1	1	$\ldots$	1	b_{2^n-1}

We have 2^n possible values and each b_i is the value of the functions, they're just bits ($b_i \in \{0,1\}$).

Observation: If $g(t_1, \ldots, t_n) = 0$ for all $(t_1, \ldots, t_n) \in \{0,1\}^n$ (g always 0), we can take $\bot = F$

$$\hat{F} = \hat{\bot} = 0 = g$$

Otherwise, at least a bit is 1, let $I \subseteq \{0, 1, \ldots, 2^n - 1\}$ be defined by $i \in I$ iff $b_i = 1$, by our earlier observation $I \neq \emptyset$.

For all $i \in I$ let us consider the i-th row of the table:

$$a_{i1}\ a_{i2} \ldots a_{in}\ |b_i$$

and $b_i = 1$ since $i \in I$.

For each $j \in \{1, \ldots, n\}$ we set

$$\begin{cases} A_{ij} := p_j & \text{if } a_{ij} = 1 \\ A_{ij} := \neg p_j & \text{if } a_{ij} = 0 \end{cases}$$

We set

$$F := \bigvee_{i \in I} \bigwedge_{j=1}^{n} A_{ij}$$

Note $F \in \mathcal{F}_{\mathcal{L}}^{(n)}$.

We have to prove that $\hat{F} = g$.

It's enough to prove that:

$$\left(\widehat{\bigwedge_{j=1}^{n} A_{ij}}\right)(t_1, \ldots, t_n = 1)$$

iff

$$(t_1, \ldots, t_n) = (a_{i1}, a_{i2}, \ldots, a_{in})$$

As defined, this formula identifies the i-th row, evaluates to 1 just on the i-th row call it (†).

Proof of (†): Clearly

$$(A_{i1} \wedge A_{i2} \wedge \ldots \wedge A_{in})\,(a_{i1}, a_{i2}, \ldots, a_{in}) = 1$$

For convenience:

$$(A_{i1} \wedge A_{i2} \wedge \ldots \wedge A_{in}) = \bigwedge_{j=1}^{n} A_{ij}$$

Since, by definition of A_{ij}

$$\hat{A}ij(a_{i1}, a_{i2}, \ldots, a_{in}) = \begin{cases} a_{ij} & \text{if } a_{ij} = 1 \\ 1 - a_{ij} & \text{if } a_{ij} = 0 \end{cases}$$

That is, $\hat{A}_{ij}(a_{i1}, \ldots, a_{in}) = 1$.
So

$$\widehat{\bigwedge} j = 1^n A_{ij}(a_{i1}, a_{i2}, \ldots, a_{in}) = 1$$

By definition of $I_\wedge$.

Let now us take $(t_1, \ldots, t_n) \in \{0,1\}^n$, with $(t_1, \ldots, t_n) \neq (a_{i1}, a_{i2}, \ldots, a_{in})$ (so $(t_1, \ldots, t_n)$ is not the ith row).

Let $j \in \{1, \ldots, n\}$ be such that $t_j \neq a_{ij}$ (there must be at least one position in which they are different, otherwise they would be the same row)

$$\begin{cases} a_{ij} = 1 & \text{iff } t_j = 0 \\ a_{ij} = 0 & \text{iff } t_j = 1 \end{cases}$$

then

$$\hat{A}_{ij}(t_1, \ldots, t_n) = \begin{cases} 1 - a_{ij} & \text{iff } a_{ij} = 1 \\ a_{ij} & \text{iff } a_{ij} = 0 \end{cases} = 0$$

So

$$\widehat{\bigwedge}_{j=1}^{n} = 0$$

By definition of $I_\wedge$. So we proved (†).

So, by (†)

$$\left(\bigwedge_{j=1}^{n} A_{ij}\right)(t_1, \ldots, t_n) = 1 \text{ iff } t_1, \ldots, t_n \text{ is the } i\text{th row}$$

So

$$\left(\bigvee_{i\in I}\overbrace{\bigwedge_{j=1}^{n}}A_{ij}\right)(t_1,\,\ldots,t_n)=1$$

iff (by definition if $I_\vee$) $(t_1,\,\ldots,t_n)$ is the ith row for some $i\in I$

iff (by definition of I) $g(t_1,\,\ldots,t_n)=1$.

So we have concluded that F, defined as:

$$F := \bigvee_{i \in I} \bigwedge_{j=1}^{n} A_{ij}$$

is such that

$$\hat{F}(t_1, \dots, t_n) = g(t_1, \dots t_n)$$

for all $(t_1, \dots t_n) \in \{0,1\}^n$, so

$$\hat{F} = g$$

□

Exercise: prove that for any $f : \{\}^n \to \{0,1\}$ there is a formula $G \in \mathcal{F}_{\mathcal{L}}^{(n)}$ such that $\hat{G} = f$ and

$$G = \bigwedge_{i \in J} \bigvee_{j=1}^{n} B_{ij}$$

where $J \subseteq \{0, \dots, 2^n - 1\}$ such that $i \in J$ iff $b_j = 0$ and

$$\begin{cases} B_{ij} = \neg p_j & \text{if } a_{ij} = 1 \\ B_{ij} = p_j & \text{if } a_{ij} = 0 \end{cases}$$

Dual of before, just switch everything.

The **functional completeness theorem** is a typical **normal form theorem**. We have two "flavors", **disjunctive and conjunctive normal form** (the two dual thingies proven, respectively).

Given $f : \{0,1\}^n \to \{0,1\}$:

$$F = \bigvee_{i \in I} \bigwedge_{j=1}^{n} A_{ij} \quad G = \bigwedge_{i \in J} \bigvee_{j=1}^{n} B_{ij}$$

Are such that $\hat{F} = f = \hat{G}$.

For $A \in \mathcal{F}_{\mathcal{L}}^{(n)}$ then there are $F_A \vee \wedge A_{ij}$ and $G_A \wedge \vee A_{ij}$ such that

$$\hat{F}_A = \hat{A} = \hat{G}_A$$

it's the same as saying

$$[F_A]_{\equiv} = [A]_{\equiv}[G_A] \equiv \Longrightarrow$$
$$\Longrightarrow \; F_A \equiv A \equiv G_A$$

Normal form: a way of picking a "standard" element in every one of the equivalence classes.

Observation: The functional completeness theorem **does not hold** (for non classical logic, but also) **for** the **set of all formulas** $\mathcal{F}_{\mathcal{L}}$. It **works only when we restrict** to formulas and functions on **the first** n **propositional letters**, for any $n \in \mathbb{N}$

$$Func^{(n)} = \hat{\mathcal{F}}_{\mathcal{L}}^{(n)} \cong {}^{\mathcal{F}_{\mathcal{L}}^{(n)}}/_{\equiv}$$

Functional completeness theorem proves the first $=$.

If we consider all propositional letters (remember $|\mathcal{L}| = \aleph_0$)

$$Func^{(\omega)} = \{f : \{0,1\}^\omega \to \{0,1\}\}$$

But $\{0,1\}^\omega$ is the set of all infinitely long sequences of bits.

$$|Func^{(\omega)}| = |[0,1] \cap \mathbb{R}| = |\mathbb{R}| = 2^{\aleph_0} = \aleph_1 > \aleph_0 = |\mathcal{F}_\mathcal{L}/\equiv|$$

The normal forms:

- **Conjunctive**

$$\bigwedge_{i \in J} \bigvee_{j=1}^{n} B_{ij}$$

- **Disjunctive**

$$\bigvee_{i \in I} \bigwedge_{j=1}^{n} A_{ij}$$

Are said **canonical** or **complete**, conjunctive/disjunctive normal forms, that is every one of the disjuncts/conjuncts **mentions all the letters** $p_1, p_2, \ldots, p_n$.

3.13 Functionally complete set of connectives

A **set** $C \subseteq \{\wedge, \vee, \to, \neg, \leftrightarrow, \top, \bot, \ldots\}$ (subset of connectives) is **functionally complete** iff for all $F \in \mathcal{F}_\mathcal{L}^{(n)}$ (for all $n \in \mathbb{N}$) there is some $G \in \mathcal{F}_\mathcal{L}^{(n)}$ written using only the connectives in C such that $\hat{F} = \hat{G}$ (or $F \equiv G$, or $[F]_\equiv = [G]_\equiv$, or $\models F \leftrightarrow G$, they're all equivalent).

The set

$$\{\wedge, \vee, \neg\}$$

is **functionally complete**, by the **functionally completeness theorem**.

Other functionally complete sets:

- $\{\neg, \wedge\}$: same as above, use De Morgan laws to obtain this set
- $\{\neg, \vee\}$: also De Morgan
- $\{\neg, \to\}$: can be proven remembering that $A \wedge B \equiv (\neg A) \to B$
- $\{\to, \bot\}$: equivalent to $\neg A \equiv A \to \bot$

NAND: Operator defined as negation of AND, it's usually defined as:

$$A \text{ nand } B := \neg(A \wedge B)$$

$$I_{\text{ nand }} = \{0,1\}^2 \to \{0,1\}$$

$$I_{\text{ nand }}(x,y) = 1 - \min\{x,y\}$$

A	B	$\neg$	$(A \wedge B)$	nand
0	0	1	0	1
0	1	1	0	1
1	0	1	0	1
1	1	0	1	0

Recalling that $\{\neg, \wedge\}$ is functionally complete, the set only composed by **nand** is **functionally complete**:

$$\begin{cases} \neg A \equiv A \text{ nand } A \\ A \wedge B \equiv \neg(A \text{ nand } B) \equiv (A \text{ nand } B) \text{ nand } (A \text{ nand } B) \end{cases}$$

Exercise: prove that nor is functionally complete, (A nor $B \equiv \neg(A \wedge B)$).

Exercise: are

- $\{\wedge, \rightarrow\}$
- $\{\wedge, \vee\}$
- $\{\neg, \bot\}$

functionally complete?

They're not, there are functions that cannot be expressed with these sets.

The last one doesn't have any binary operators, so it can't be functionally complete, how do I express binary operators? You can't build functions depending on more that one variable.

While for the first two, we don't have the negation, we just have "increasing" functions.

So we can't express *everything*, for example a formula

$$F(1, 1, \ldots, 1) = 0$$

is impossible, since we can't produce a zero starting from ones only.

3.14 Normal Forms

We have seen canonical/complete normal forms (CNF, DNF) and they are **generally too long**. We want to produce CNFs and DNFs without their being canonical, in order to obtain shorter formulas.

3.14.1 Implication Free Normal Form IFNF:

We're getting rid of implication.

Definition: a Formula $F \in \mathcal{F}_{\mathcal{L}}$ is in IFNF ($F \in IFNF$) **iff** F **does not contain** any occurrence **of** $\rightarrow$.

To do that we just replace every $\rightarrow$ (we've seen it can be replaced, we'll see some transformations below).

3.14.2 Negation Normal Form NNF

Definition: A formula $F \in IFNF$ is in NNF ($F \in NNF$) iff the **negation connective** is applied **only to propositional letters** (if $\neg B$ is a sub-formula of F, then $B = p_i$ for some $p_i \in \mathcal{L}$).

Examples:

- Is $p_1 \wedge (p_2 \vee p_3$ in NNF? Obviously no, but its equivalent $p_1 \wedge (\neg p_2 \wedge \neg p_3$
- Is $p_1 \rightarrow \neg p_2$ in NNF? No, since it's not $\in IFNF$ (the negation is only on letters though)

We have "subsets" of Normal Forms:

$$\begin{matrix} CNF & \subseteq \\ DNF & \subseteq \end{matrix} NNF \subseteq IFNF$$

Notation: $G \preceq F$, G is a **Subformula** of F (for $G, F \in \mathcal{F}_\mathcal{L}$) if G occurs in every L-construction of F (you need G to build F, get the minimal L-construction and check if G is in there).

Transformations: We shall use the following equivalences to **replace subformulas** with **equivalent ones**:

1. $C \to D \equiv (\neg C) \vee D$
2. $\neg\neg C \equiv C$
3. $\neg(C \vee D) \equiv \neg C \wedge \neg D$ (De Morgan)
4. $\neg(C \wedge D) \equiv \neg C \vee \neg D$ (De Morgan)

We shall use these logical equivalences to transform subformulas of a given formula $F \in \mathcal{F}_\mathcal{L}$ using these from left to right, we're **using them as transformation rules** (we could do the opposite, but goes against our normal forms).

Lemma: For any $F \in \mathcal{F}_\mathcal{L}$ we can build another formula $G \in \mathcal{F}_\mathcal{L}$ (in finite number of steps) such that $F \equiv G$ and $G \in IFNF$. Any formula can be "transformed" into an equivalent one in IFNF.

Proof. For any $E \in \mathcal{F}_\mathcal{L}$ we introduce an index of complexity, every step it decreases and it goes at 0 only when $E \in IFNF$.

Let $i(E)$ be the number of occurrences of $\to$ in E. We can observe that $E \in IFNF$ iff $i(E) = 0$.
If $E \notin IFNF$ then $i(E) > 0$.

Take $C \to D \preceq E$ (which exists for any E with $i(E) > 0$).

Replace $C \to D$ with $(\neg C) \vee D$ and call E' the formula obtained.

$$E' = E\left[{}^{(\neg C) \vee D}/_{C \to D}\right]$$

(with a small abuse of notation, I think we defined the substitution only for letters but I think it's admissible).

Now $E \equiv E'$ for the transformation (1) and substitution theorem and

$$i(E) > i(E')$$

since we removed at least one implication.

So, given in input $F \in \mathcal{F}_\mathcal{L}$ with $F \notin IFNF$, we build a sequence

$$(\dagger) \qquad F = F_0 \equiv F_1 \equiv \ldots \equiv F_u \in IFNF$$

and

$$i(F_0) > i(F_1) > \ldots > i(F_n) = 0$$

Since we're strictly descending natural numbers since each F_{i+1} is obtained from F_i by substituting some $C \to D \preceq F$ with $(\neg C) \vee D$.

Trivially, the sequence (†) ends in a finite number of steps u with $u \leq i(F_0) = i(F)$, producing a formula F_u such that $F_u \equiv F$, $F_u \in IFNF$.

□

Lemma: for any $F \in IFNF$ we can build, in a finite number of steps, a **sequence**

$$F = F_0 \equiv F_1 \equiv \ldots \equiv F_v = G$$

such that $G \equiv F$ **and** $G \in NNF$.
This is *kinda* the same as before, but with the NNF. Starting from a formula $\in IFNF$ we can transform it into NNF.

The **measure of complexity** is not as simple, we can't just *count the instances* of $\neg$, usually the raw number increases but the negation goes closer to the propositional letters.

For any $A \in IFNF$ let $\ell(A) :=$ the **number of** occurrences of **connectives in** $\{\wedge, \vee, \neg\}$ in A.

For any $E \in IFNF$ let

$$m(E) := \sum_{\neg A \preceq E} \ell(A)$$

So $m(E) = 0$ iff $E \in NNF$.

If $E \in NNF$ then $\neg A \preceq E$ iff $A = p$ for some $p \in \mathcal{L}$, so if A is a propositional letter, it contains no connectives $\ell(A) = 0$, so $m(E) = \sum_{\neg A \preceq E} 0 = 0$.

if $m(E) = 0$ then $\sum_{\neg A \preceq E} 0$ then $\ell(A) = 0$ for all $\neg A \preceq E$, then $A = p$ for some $p \in \mathcal{L}$, then $E \in NNF$ (by definition of NNF).

I don't really know what happened here, the definition of m is still a bit of mystery to me. I would really appreciate a simple explanation.

We have to show that each application of our second, third and fourth transformation is such that one step from F_i to F_{i+1} has $m(F_i) > m(F_{i+1})$.

Cases:

- we use $\neg\neg C \preceq F_i$, so we need to prove

  $$m(\neg\neg C) > m(C)$$

  By definition:

  $$m(C) = \sum_{\neg A \preceq C} \ell(A)$$

$$m(\neg\neg C) = \sum_{\neg A \preceq \neg\neg C} \ell(A) = \ell(\neg C) + \sum_{\neg A \preceq \neg C} \ell(A) =$$
$$= \ell(C) + 1 + \ell(C) + \sum_{\neg A \preceq C} \ell(A)$$

So $m(\neg\neg C) > m(C)$ by at least 1.

- we use $\neg(C \wedge D) \preceq F_i$, we need to prove:

$$m(\neg(C \wedge D)) > m(\neg C \vee \neg D)$$

So

$$m(\neg(C \wedge D)) = \sum_{\neg A \preceq \neg(C \wedge D)} \ell(A) = \ell(C \wedge D) + \sum_{\neg A \preceq C} \ell(A) + \sum_{\neg A \preceq D} \ell(A) =$$

$$= \ell(C) + 1 + \ell(D) + \sum_{\neg A \preceq C} \ell(A) + \sum_{\neg A \preceq D} \ell(A)$$

And

$$m(\neg C \vee \neg D) = \sum_{\neg A \preceq \neg C \vee \neg D} \ell(A) = \ell(C) + \ell(D) + \sum_{\neg A \preceq C} \ell(A) + \sum_{\neg A \preceq D} \ell(A)$$

then $m(\neg(C \wedge D)) > m(\neg C \vee \neg D)$.

- we use $\neg(C \vee D) \preceq F_i$, we need to prove:

$$m(\neg(C \vee D)) > m(\neg C \wedge \neg D)$$

Same as before, just switch symbols.

Observe: if $G \preceq F$, going from G to H and subsequently from F to F', $F' = F[^H/_G]$

$$\text{if } \; m(G) > m(H) \; \text{ then } \; m(F) > m(F')$$

As a matter of fact

$$m(F) = \sum_{\neg A \preceq F} \ell(A) = \sum_{\neg A \preceq G} \ell(A) + \sum_{\neg A \preceq F, \neg A \not\preceq G} \ell(A) = m(G) + \sum_{\neg A \preceq F, \neg A \not\preceq G} \ell(A)$$

$$m(F') = \sum_{\neg A \preceq F'} \ell(A) = \sum_{\neg A \preceq H} \ell(A) + \sum_{\neg A \preceq F, \neg A \not\preceq H} \ell(A) = m(H) + \sum_{\neg A \preceq F, \neg A \not\preceq G} \ell(A)$$

$$\Longrightarrow \; m(F) > m(F')$$

We have **proved that** if $F \in IFNF$ we can build a sequence

$$F = F_0 \equiv F_1 \equiv \ldots \equiv F_v = G$$

such that $G \in NNF$ and $G \equiv F$ since

$$m(F_0) > m(F_1) > \ldots > m(F_v) = 0$$

The **number of steps** is **at most** the **initial complexity** of F, $m(F) = m(F_0)$. Which scales the complexity in a linear fashion.

We have, by the proofs of the previous Lemmas, **efficient** (linear time) **algorithms** to put a **formula** $F \in \mathcal{F}_{\mathcal{L}}$ into an **equivalent one in IFNF and** then **NNF**.

F can be efficiently transformed in an $F^{IF} \in IFNF$, which can be transformed, still efficiently, in an $F^{N} \in NNF$, taking at most $i(F) + m(F)$ steps.

The IF and N mean, respectively, that the formula is $\in IFNF$ and $\in NNF$.

3.14.3 Conjunctive Normal Form CNF

But I hear you saying "*we've already seen this*", do we have to look at it again? We already know the *canonical* CNF and DNF but now they're the *star of the show*, so let them shine (this gives "What about second breakfast?" vibes).

Terminology:

- **Definition: Literal**, by literal we mean a formula in the form p or $\neg p$ for some $p \in \mathbb{L}$. A letter or a negation of one.

- **Definition: Opposite Literal**, if A is a literal, then we denote by $\overline{A}$, called opposite of A, the literal so defined:

$$\begin{cases} \text{if } \ A = p & \text{then } \overline{A} = \neg p \\ \text{if } \ A = \neg p & \text{then } \overline{A} = p \end{cases}$$

  Equivalently

$$\overline{A} := (\neg A)^N$$

  An opposite literal is the opposite of a literal, do you really need a definition?

- **Definition: Clause**, a clause is a disjunction of a finite number of literals:

$$l_1 \vee l_2 \vee \ \ldots \ \vee l_k$$

  where each l_i is a literal.

- **Definition: CNF**, a CNF is a conjunction of a finite number of clauses:

$$C_1 \wedge C_2 \wedge \ \ldots \ \wedge C_n$$

  where each C_j is a clause.

- **Definition: Empty Clause**, an empty clause ($\square$) is, by definition, a disjunction of zero literals.

- **Definition: Empty CNF**, the empty CNF ($\emptyset$) is the conjunction of zero clauses. Very much different from the last one.

Thinking about l_1 as a literal, and the disjunctions

$$l_1$$

$$l_1 \vee l_2$$

$$l_1 \vee l_2 \vee l_3$$

and so on

$$l_1 \vee l_2 \vee l_3 \vee \ \ldots \ l_k \vee \ \ldots$$

If any of these is satisfiable, then all of the ones below are satisfiable, so it makes sense to put $\square$ on top of the first literal l_1, so, by definition, the empty literal $\square$ **is unsatisfiable and a**

contradiction.

Same game with clauses and empty CNF $\emptyset$:

$$C_1$$

$$C_1 \wedge C_2$$

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n \wedge \ldots$$

But here the satisfiability flows upwards, if one is satisfied then all of the above are satisfied. The empty CNF $\emptyset$ is still above all, and as such is satisfied by all assignments and is a tautology

3.14.4 Disjunctive Normal Form DNF

A DNF is a disjunction of a finite number of formulas:

$$D_1 \vee D_2 \vee \ldots \vee D_h$$

where each D_i (dual of a clause, hasn't got a proper name, maybe co-clause) is a conjunction of a finite number of literals (or the dual of a literal, but it's still a literal):

$$l_1 \wedge l_2 \wedge \ldots \wedge l_m$$

Every definition stated before has a DNF dual but they don't have a proper name since *nobody cares*.

Lemma: Every formula $F \in \mathcal{F}_{\mathcal{L}}$ is **logically equivalent to** a formula $F^C \in CNF$ and to a formula $F^D \in DNF$

$$(CNF \ni F^C \equiv F \equiv F^D \in DNF)$$

Proof. By structural induction of F:

- **Base**: $F = p$, $p \in \mathcal{L}$ then $p^C \equiv p \equiv p^D$. A single literal is already in CNF and DNF, $p^C = p = p_D$.
- **Step**: the case $F = \neg G$, by I.H., there are $G^c \in CNF$, $G^D \in DNF$ with $G^C \equiv G \equiv F^D$

  $$F = \neg G \equiv (\neg(G^D))^N \in CNF$$

  $$F = \neg G$$
  $$\neg(G^D) = \neg(D_1 \vee D_2 \vee \ldots \vee D_v)$$

  with each D_i
  $$D_i = l_{i1} \wedge l_{i2} \wedge \ldots \wedge l_{iq}$$

  but by De Morgan
  $$\equiv \neg D_1 \wedge \neg D_2 \wedge \ldots \wedge \neg D_v$$

  And, still by De Morgan, each $\neg D_i$
  $$\equiv (\bar{l}_{i1} \vee \bar{l}_{i2} \vee \ldots \vee \bar{l}_{iu_i})$$

So $(\neg(G^D))^N$ is the conjunction $\neg D_1 \neg D_2 \wedge \ldots \neg D_v$ of a finite number of formulas $(\neg D_i)$, each one equivalent to a disjunction of a finite number of literals, that is $(\neg(G^D))^N \in CNF$, and $F \equiv \neg G \equiv (\neg(G^D))^N$.

□

Exercise: prove that if $F = \neg G$ then

$$F \equiv (\neg(G^C))^N, \quad (\neg(G^C))^N \in DNF$$

For the **other connectives**:

Proof. Case: $F = (G \wedge H)$, by I.H.

$$G^C, H^C \in CNF, \quad G^D, H^D \in DNF$$

$$\text{with } G^C \equiv G \equiv G^D, \;\; H^C \equiv H \equiv H^D$$

We want to build **2 new formulas**:

- $F \equiv F^D$, defining F^D as $:= G^D \vee H^D \in DNF$, then by definition is $\in DNF$, and by construction $G^D \vee H^D \equiv F$. Disjunction of disjunctions of finitely many co-clauses, so still $\in DNF$.

- $F \equiv F^C$, then

  $$G^C = G_1 \wedge G_2 \wedge \ldots \wedge G_u$$
  $$H^C = H_1 \wedge H_2 \wedge \ldots \wedge H_v$$

  So

  $$G \vee H \equiv G^C \vee H^C \equiv (G_1 \wedge G_2 \wedge \ldots \wedge G_u) \vee (H_1 \wedge H_2 \wedge \ldots \wedge H_v) \equiv$$

  Using generalized distributivity:

  $$\equiv \bigwedge_{i=1}^{u} \bigwedge_{j=1}^{v} (G_i \wedge H_j)$$

Each G_i and each H_j are clauses, i.e., disjunctions of finitely many literals, so each term in the form $G_i \vee H_j$ is again a clause, and then the whole formula is a finite conjunction of clauses and, by definition, a CNF, also $\equiv G \vee H = F$.

□

Then we have the case $(G \wedge H)$, which is left as an exercise to the reader.

Case (to complete our set of connectives): $(G \to H)$, which is also left as an exercise, because implication can be reduced to $G \to H \equiv (\neg G) \vee H$.

Transforming example:

$$p_1, p_2, p_3, q_1, q_2, q_3 \in \mathcal{L}$$

Then

$$(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee (p_3 \wedge q_3) \in DNF$$

We want to **convert** these 3 disjuncts of 2 literals each **to CNF**. We want to distribute the first conjunction over the rest:

$$\equiv [p_1 \vee ((p_2 \wedge q_2) \vee (p_3 \wedge q_3))] \wedge [p_2 \vee ((p_2 \wedge q_2) \vee (p_3 \wedge q_3))]$$

Now we do the same thing with the second conjunction

$$\equiv (p_1 \vee p_2 \vee (p_3 \wedge q_3)) \wedge (p_1 \vee q_2 \vee (p_3 \wedge q_3)) \wedge (q_1 \vee p_2 \vee (p_3 \wedge q_3)) \vee (p_1 \vee q_2 \vee (p_3 \wedge q_3))$$

Distributivity:

$$\equiv (p_1 \vee p_2 \vee p_3) \wedge (p_1 \vee p_2 \vee q_3) \wedge (p_1 \vee q_2 \vee p_3) \wedge (p_1 \vee q_2 \vee q_3) \wedge$$

$$\wedge (q_1 \vee p_2 \vee p_3) \wedge (q_1 \vee p_2 \vee q_3) \wedge (q_1 \vee q_2 \vee p_3) \wedge (q_1 \vee q_2 \vee q_3)$$

And this is $\in CNF$, having 8 conjuncts of 3 literals each.

Exercise: generalize to every $n \in \mathbb{N}$.

A DNF of n disjuncts of 2 literals each, becomes a CNF with 2^n c+onjuncts with n literals each.

Doing the inverse (CNF to DNF) has the same exponential dilation.

This page is intentionally left blank.

※ Notions of computational complexity

We want to reason about the **complexity of a decision problem** (has an answer which is either *yes* or *no*).

Definition: A (decision) **problem** (or "*language*") is a subset $L \subseteq \Sigma^*$, i.e., a subset L of the set of all finite strings (words) over a finite alphabet (set) Σ.

For instance, formulas of $\mathcal{F}_{\mathcal{L}}$ can be written over the alphabet

$$\Sigma = \{\wedge, \vee, \neg, \rightarrow\} \cup \{), (\} \cup \{p, |\} \quad p_i \in \mathcal{L}$$

So we can say that:

$$\mathcal{F}_{\mathcal{L}} \subseteq \Sigma^*$$

Given a $w \in \Sigma^*$, to know if it belongs to $\mathcal{F}_{\mathcal{L}}$, so $w \in \mathcal{F}_{\mathcal{L}}$, then **iff** w is a formula, there is an L-construction for w.

There are other decision problems we've seen, as defined:

$$NNF \subseteq \Sigma^*, \quad CNF \subseteq \Sigma^*, \quad DNF \subseteq \Sigma^*$$

These happen to be **syntactical problem** which are (*usually*) **easy**.

But there are **semantics problems**, which are (*usually*) **not so easy**. For instance:

$$SAT \subseteq \Sigma^*$$

Input $w \in \Sigma^*$, definition: $w \in SAT$ iff $w \in \mathcal{F}_{\mathcal{L}}$ and w is satisfiable.
We can decide if a formula is satisfiable, but it's long (think of a truth table, it's exponential in length).

Another semantical problem:

$$TAUTO \subseteq \Sigma^*$$

Same as before, but all entries must be 1, not at least one.

Same with checking if a formula is unsatisfiable:

$$UNSAT \subseteq \Sigma^*$$

and complement of SAT, which is checking if it's unsatisfiable but it doesn't have to be a formula

$$SAT^C \subseteq \Sigma^*$$

We can solve these problems, although inefficiently.

4.1 Computational Model

We want to **fix a computational model** which will provide a reasonable notion of **elementary step of computation**.

Church-Turing (extended) Thesis: There are **many** different reasonable **computational models**, but the **set of computable functions** (what can actually be computed on these models) **is always the same**.
Every "reasonable" computational model is "equivalent", in the sense that every one of these models defines the same set of efficiently computable functions (decidable problems).

We choose the **model of Turing Machines**, which comes in several flavors:

- Vanilla: **Deterministic Turing Machines DTM**
- Chocolate: **Non-Deterministic Turing machines NDTM**

Which will be useful to define complexity classes.

I'm not going to describe a Turing machine, you got here, you know Turing machines, always the same quintuple (q, b, q', b', s).

The computation goes on until it reaches a state declared (in the definition of the machine) to be final, and accepts the input if this final state is declared to be accepting.
The machine could also never stop, thus not accepting the input, and it could also explicitly reject the input.

Distinction between **deterministic** (vanilla) and **non-deterministic** (chocolate) Turing machines:

- **Deterministic** (called DTM): means that for **any state-symbol pair** (q, b) there is **at most one instruction** of the form $q, b \to \ldots$.

- **Non-Deterministic** (called NDTM): there may be **several distinct instruction for any state-symbol pair** of the form $q, b \to \ldots$; several instruction for the same state and symbol. When such an instruction is reached, there's a universe for each possible instruction the machine can execute. A DTM is a sequence of steps, a NDTM is a tree of possibilities and an input is accepted when at least one of the leafs in the NDTM tree reaches an accepting state.

4.2 Computational Complexity Classes

Decision problems which are **efficiently** decidable.

Class $\mathcal{P}$: problems decidable in deterministic polynomial time.

A problem $L \subseteq \Sigma^*$ is in $\mathcal{P}$ ($L \in \mathcal{P}$) **iff** there exists a Deterministic Turing Machine T and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for each possible input $w \in \Sigma^*$

- the computation of T on input w ($T(w)$) ends within $p(||w||)$ steps (with $||w||$ meaning the length of w, the number of characters composing w).
- for each $w \in L$, $T(w)$ accepts. For each $w \notin L$, $T(w)$ does not accept.

A polynomial p is a polynomial if it can be expressed as a polynomial.

Class $\mathcal{NP}$: problems decidable in non-deterministic polynomial time.

Notion of Efficiency

The **composition of polynomials is a polynomial**, so multiple polynomials together are still a polynomial and thus efficient (but with a **higher degree**, useful for combining algorithms).
Composition of polynomials of a fixed degree (if greater than 1) is not a polynomial of that degree.

Multiplication of polynomials is, again, **a polynomial**, but, again, the result is of a **different degree**.

We want that our notion of "**efficient computation**" to be **robust under composition, multiplication and change of the computational model**. So we're lead to **use polynomial** (with no fixed bound on the degree).

TL;DR: polynomials are efficient.

Previously seen syntactical problems are $\in \mathcal{P}$.

Class $\mathcal{NP}$: Like P but on NDTM, a problem $L \subseteq \Sigma^*$ is $\in \mathcal{NP}$ **iff** there exists a deterministic Turing Machine T and two polynomials $p, q : \mathbb{N} \to \mathbb{N}$ such that for all $w \in \Sigma^*$ and each $z \in \Gamma^*$ (Γ may be the same as Σ) we have:

- $T(w, z)$ ends within $p(||w||)$ steps
- For each $w \in L$ there is additional information $z \in \Gamma^*$ such that $||z|| \leq q(||w||)$ (is polynomial with respect to the length of w) and $T(w, z)$ accepts
- For each $w \notin L$ and for all $z \in \Gamma^*$ such that $||z|| \leq q(||w||)$, $T(w, z)$ does not accept

Equivalently: A problem L is in $\mathcal{NP}$ iff there **exists a NDTM** T and a **polynomial** $p : \mathbb{N} \to \mathbb{N}$ such that $w \in L$ iff $T(w)$ accepts in a number of steps $\leq p(||w||)$.

The class $\mathcal{NP}$ as the class of **polynomially verifiable guess and check procedures**: if $w \in L$, guess an information $z \in \Gamma^*$ witnessing that w is actually in L. Basically, class of problems in which, given an answer I can check that said answer confirms that $w \in L$.
Check in deterministic polynomial time that z witnesses that $w \in L$.

Example: SAT, taking $w \in \Sigma^*$ with the alphabet:

$$\Sigma = \{\wedge, \vee, \neg, \rightarrow\} \cup \{), (\} \cup \{p, |\} \quad p_i \in \mathcal{L}$$

$w \in SAT$ **iff** $w \in fl$ and w is **satisfiable**.

Is $SAT \in \mathcal{P}$? Nobody knows (since nobody knows whether $\mathcal{P} = \mathcal{NP}$, we can guess no).

Is $SAT \in \mathcal{NP}$? Looking at the definitions given for the class, can I guess and check? If a relevant truth assignment *descended from above*, could I **check that assignment in polynomial time**? Of course I can. We can guess a (restricted, obviously) assignment and check in deterministic polynomial time in $||w||$ that the assignment equals 1 (satisfies satisfiability).

Clearly $\mathcal{P} \subseteq \mathcal{NP}$.
What about the other way around? It would mean that $\mathcal{P} = \mathcal{NP}$, and it's part of the millennium problems. You probably ain't solving that.
What about **other problems**?

$UNSAT \in co - \mathcal{NP}$, i.e., it's in the complement of $\mathcal{NP}$, the set of problems for which I can't guess and check. I can't say that something is $UNSAT$ only from a random assignment.
Same for $SAT \in co - \mathcal{NP}$, $TAUTO \in co - \mathcal{NP}$ (since we can just check that the negation of the input is unsat).

On $\mathcal{NP}$ **problems** we can do **guess and check if an answer is correct**, while on $co - \mathcal{NP}$ **problems** we can't do that, equivalently we can **guess and check if the answer is NOT correct**.

$\mathcal{NP}$-complete ($\mathcal{NP}c \subseteq \mathcal{NP}$): we say that L is $\mathcal{NP}c$ **iff**

- $L \in \mathcal{NP}$
- Every $L' \in \mathcal{NP}$ is such that L' is **polynomially reducible to** L (we say that L is $\mathcal{NP}$-hard)

4.3 Polynomially reducible

A problem $L_1 \subseteq \Sigma^*$ is **polynomially reducible** to a problem $L_2 \subseteq \Gamma^*$ **iff** there exists a **DTM** T_{L_1,L_2} and a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$ such that for all $w \in \Sigma^*$, T_{L_1,L_2} transforms w into $w' \in \Gamma^*$ using a number of steps $\leq p(||w||)$ and such that $w \in L_1$ iff $w' \in L_2$. We write $L_1 \preceq_p L_2$.

This means that solving problem L_2 means solving L_1, each input of one problem can be **efficiently** (in polynomial time) **transformed** into an instance of the other.

If $L \in \mathcal{NP}c$ and $L \in \mathcal{P}$ then $\mathcal{NP} = \mathcal{P}$, since every other problem $\in \mathcal{NP}$ could be reduced to a problem $\mathcal{NP}c$, making $\mathcal{P}$ and $\mathcal{NP}$ collapse.

4.3.1 Cook-Levin Theorem

Basically, it says that there are $\mathcal{NP}c$ problems.

Cook proved this with $CNF - SAT$, so $CNF - SAT \in \mathcal{NP}c$ (and thus SAT, since it's the *deluxe extended version*).

Polynomial reduction **example:**

- Tauto and UNSAT

$$TAUTO \preceq_p UNSAT$$

$$w \mapsto \neg w$$

$$w \in TAUTO \text{ iff } \neg w \in UNSAT$$

  C'mon, flip the input and it's the same.

- CNFSAT to SAT: $w \in \Sigma^*$, $w \in CNFSAT$ iff $w \in CNF$ and $w \in SAT$

$$CNFSAT \preceq_p SAT$$

$$w \mapsto \begin{cases} w & \text{if } w \in CNF \\ \bot & \text{if } w \notin CNF \end{cases}$$

By Cook-Levin Theorem $CNFSAT \preceq_p SAT$ and $SAT \preceq_p CNFSAT$.

The proof given by Cook of the Cook-Levin theorem is by "simulation".

Given a problem $L \subseteq \Sigma^*$, $L \subseteq \mathcal{NP}$, we have to show that $L \preceq_p CNFSAT$. By def of $\mathcal{NP}$ there is a NDTM T_L deciding L in non-deterministic polynomial time.

The idea of the proof is, for each $w \in \Sigma^*$ to build a CNF which mimics the computation of T_L on w.
Finally, the produced CNF is satisfiable **iff** $T_L(w)$ accepts.

Since T_L works in polynomial time with respect to $||w||$, a careful analysis of the produced CNF shows that its length is polynomial with respect to $||w||$.

Cook-Levin theorem **shows that** $L \preceq_p CNFSAT$ **for any** $L \in \mathcal{NP}$ (equivalent to saying that $CNFSAT \in \mathcal{NP}c$).

There's a big graph of complexity and computability classes here, *I ain't doing all that.*

Some **problems** and their **relative classes**

- $CNFSAT \preceq_p DNFUNTAUTO \in \mathcal{NP}c$.
- $CNFTAUTO \preceq_p DNFUNSAT \in \mathcal{P}$.
- $CNFUNTAUTO \preceq_p DNFSAT \in \mathcal{P}$.
- $DNFTAUTO \preceq_p CNFUNSAT \in co - \mathcal{NP}c$.

Questions/Observations:

- Is it computationally efficient transforming a CNF into a logically equivalent DNF? It's *very unlikey* since it would means reducing CNFSAT to DNFSAT efficiently thus making $\mathcal{P} = \mathcal{NP}$. $CNFSAT \preceq_p DNFSAT \implies \mathcal{P} = \mathcal{NP}$.
- Is it computationally efficient transforming a DNF into a logically equivalent CNF? No, same as above but with $\mathcal{NP}c \ni DNFUNTAUTO \preceq_p CNFUNTAUTO \in \mathcal{P}$

Nonetheless, we'll show a poly reduction from SAT to CNFSAT

$$SAT \preceq_p CNFSAT$$

If $SAT \preceq_p DNFSAT$ then $\mathcal{P} = \mathcal{NP}$.

4.4 Equisatisfiability

Definition: Let $A, B \in fl$, then A is equisatisfiable with B (A and B are equisatisfiable) **iff** A is satisfiable **iff** B is satisfiable.

That is: either A and B are both satisfiable xor A and B are both unsatisfiable (might not be the same assignments, just same satisfiability level).

Equisat **considers only satisfiability**, it doesn't care about logical equivalence (but the latter always implies the former).

Is equisat an equivalence relation? We need to prove reflexivity, symmetry and transitivity.
So we can confidently say that **it's an equivalence relation**.

Is equisat a congruence with respect to the operations considered (connectives)? We can find equisat formulas not always satisfied by all the same assignments (we can prove this for any of the connectives).
So it's **not a congruence**.

Exercises:

- is equisat a congruence with respect to $\wedge$?
- is equisat a congruence with respect to $\vee$?
- is equisat a congruence with respect to $\rightarrow$?

$A \equiv B$ implies that A equisat B. $\equiv$ is, as a relation, more refined than equisat (i.e., equisat is rougher than $\equiv$).

The **equivalence classes of equisat** are **unions** of **equivalence classes of** $\equiv$, more specifically, equisat cares only between unsatisfiable and satisfiable (takes the $\perp$ class and the union of all the rest), while $\equiv$ has classes for each assignment.

4.4.1 SAT to CNFSAT

We shall **reduce** SAT polynomially $\preceq_p$ to $CNFSAT$ **using preservation of equisatisfiability** instead of preservation of logical equivalence.

Observation: Recall that $SAT \preceq_p NNFSAT$, preserving logical equivalence (since transforming a formula to NNF can be done in polynomial time while preserving logical equivalence).

Observation: If $A \in NNF$ and assume that $A \notin CNF$ then A contains at least one subformula of the following form:

$$(\dagger) \quad C \vee (D_1 \wedge D_2) \text{ or } (D_1 \wedge D_2) \vee C \quad (\dagger\dagger)$$

for some $C, D_1, D_2 \in NNF$.

Since $A \in NNF$, A is a $\wedge$-$\vee$ combination of literals. If every occurrence of $\vee$ is in a subformula with the main connective $\wedge$ then $A \in CNF$. If $A \notin CNF$ there must be a subformula of the shape $(\dagger)$ or $(\dagger\dagger)$.

We call any occurrence of $(\dagger)$ or $(\dagger\dagger)$ a "**violation**".
Preliminary step: since $(\dagger) \equiv (\dagger\dagger)$, we use substitution theorem to **replace every occurrence of the kind** $(\dagger\dagger)$ **with** $(\dagger)$. This requires constant time.

We can now assume that **every violation is in the form** $(\dagger)$.

Let $A \in NNF$ and let $B = (\dagger)$ with $B \preceq A$. Then B is a violation.

The **original formula** may contain a **finite number** n **of violations**, so we **eliminate at least one at a time**, while **maintaining equisatisfiability** and doing this in **efficient time**, to obtain something equisat.

For **each violation** B we **introduce** a new (not occurring in A) propositional **letter** $a_B \in \mathcal{L}$.

We **define**

$$B' := B\left[{}^{a_b}/_{D_1 \wedge D_2}\right] \wedge (\neg \vee D_1) \wedge (\neg a_b \vee D_2)$$

And we call the part after the first $\wedge$ "**tail**".
Where

$$B'' := \left[{}^{a_b}/_{D_1 \wedge D_2}\right]$$

is the formula obtained by replacing in B every occurrence of $D_1 \wedge D_2$ with the letter a_B.

So

$$B' = (C' \vee a_b) \wedge (\neg \vee D_1) \wedge (\neg a_b \vee D_2)$$

with

$$C' = C\left[{}^{a_b}/_{D_1 \wedge D_2}\right]$$

We shall prove that A **and** A' obtained by **replacing** B **in** A **with** B' are **equisat** (while, in general, not being $\equiv$).

Examples $p, q, r \in \mathcal{L}$ and a fresh $a \in \mathcal{L}$ $(a \neq p, a \neq q, a \neq r)$.
We can show how a simple violation can be removed:

$$p \vee (q \wedge r) \mapsto (p \vee a) \wedge (\neg \vee q) \wedge (\neg a \vee r)$$

The first is not a CNF, while the second is $\in CNF$. We now have to show that they're equisat.

With the assignment

$$v(q) = v(r) = v(a) = 1, \quad v(p) = 0$$

we can see that

$$\tilde{v}(p \vee (q \wedge r)) = 1$$
$$\tilde{v}((p \vee a) \wedge (\neg \vee q) \wedge (\neg a \vee r)) = 1$$

So they're **both satisfiable** $\implies$ they're **equisat**.

But *are they logical equivalent?* Considering

$$w(q) = w(r) = 1, \quad w(p) = w(a) = 0$$

Then

$$\tilde{w}(p \vee (q \wedge r)) = 1$$
$$\tilde{w}((p \vee a) \wedge (\neg \vee q) \wedge (\neg a \vee r)) = 0$$

So they're equisat, but **not** $\equiv$.

We **can generalize this process** with not only letters but formulas in general, composed of n literals.

Transforming **from DNF** of n disjuncts of 2 literals each **to CNF** while maintaining logical equivalence takes 2^n conjuncts, while **maintaining just equisatisfiability** only **takes** $2n+1$ (so it's **linear** instead of exponential).

Algorithm:

- Let $F \in \mathcal{F}_{\mathcal{L}}$.
- Let $A \in NNF$ such that $A \equiv F$. A is built in polynomial time with respect to $||F||$.
- Build a sequence
$$F \equiv A := A_0 \rightsquigarrow A_1 \rightsquigarrow \ \ldots \ \rightsquigarrow A_u \in CNF$$
where $u \leq$ the number of violations occurring in A ($u \leq ||A||$, max number of steps) and A_{i+1} is obtained from A_i by removing at least one violation as follows: remove a violation $B = C \vee (D_1 \wedge D_2)$ by substituting in A_i, B with B': the outcome is A_{i+1}. Then we shall guarantee that for each step A_i equisat A_{i+1}.
Note that $F \equiv A$ equisat with $A_u \in CNF$.

Notice that in passing **from** A_i **to** A_{i+1} (from B to B') we have a **dilatation of the formulas** $||A_{i+1}|| \geq ||A_i||$. But by **how much**? Remembering that

$$B \rightsquigarrow B' = B\left[{}^{a_b}/_{D_1 \wedge D_2}\right] \wedge (\neg \vee D_1) \wedge (\neg a_b \vee D_2)$$

So $||B'|| \leq ||B|| + k$ where k is a constant (12 counting all symbols).
So $||A_{i+1}|| \leq ||A_i|| + k$.

If the initial number of violations in $A = A_0$ is v, after at most $u \leq v$ steps we have produced $A_u \in CNF$, and $||A_u|| \leq ||A|| + u \cdot k \leq (k+1)||A||$. It increases by a constant number of symbols k, for at most u times.

So the **algorithm runs in deterministic polynomial time**.

Proving correctness: To prove the correctness of the algorithm it suffices to prove that **for each index** $i = 0, 1, \ldots, u-1$, A_i **equisat** A_{i+1}, then by the transitivity of equisat $F \equiv A_0$ equisat $A_u \in CNF$, F equisat A_u.

Let $B \preceq A_i$ be a violation for A. Then

$$B = C \vee (D_1 \wedge D_2)$$

$$B' = B'' \wedge (\neg a_b \vee D_1) \wedge (\neg a_B \vee D_2) \text{ with } B'' = \left[{}^{a_b}/_{D_1 \wedge D_2}\right]$$

Then we can say that

$$\begin{cases} A & := A_i\left[{}^{q}/_{B}\right] \\ A_i & := A\left[{}^{B}/_{q}\right] \\ A_{i+1} & := A\left[{}^{B'}/_{q}\right] \end{cases}$$

where q is a fresh propositional letter.

Observation: Take the tail $(\neg a_b \vee D_1) \wedge (\neg a_B \vee D_2)$, we can see that

$$(\neg a_b \vee D_1) \wedge (\neg a_B \vee D_2) \equiv a_b \rightarrow (D_1 \wedge D_2)$$

So a_B is always $\leq$ than $(D_1 \wedge D_2)$, which is sufficient, but we could force $a_B \leftrightarrow (D_1 \wedge D_2)$, but the tail becomes longer (and I don't care tbh).

Assume A_i is **satisfiable** and we prove that A_{i+1} is **satisfiable too**. Se there's at least a $v : \mathcal{L} \to \{0,1\}$ such that $\tilde{v}(A_i) = 1$ (v exists by assumption that $A_i \in SAT$).

Definition: the assignment $v_{a_B} : \mathcal{L} \to \{0,1\}$ such that it's

$$\begin{cases} v_{a_B} := v(p) & \text{for all } p \in \mathcal{L}, p \neq a_B \\ v_{a_B} := \tilde{v}(D_1 \wedge D_2) & \text{otherwise} \end{cases}$$

It's well defined.

Notice that $\tilde{v}_{a_B}(a_B \to (D_1 \wedge D_2)) = 1$, by the semantics of implication ($\tilde{v}_{a_B}$ satisfies the tail).

Then $(\dagger) = \tilde{v}_{a_B}(B') = \tilde{v}_{a_B}(B'')$, by the semantics of $\wedge$, since $B' = B''\wedge$ tail.

Now

$$\begin{cases} B & = (B\,[a_B/D_1 \wedge D_2])\,[D_1 \wedge D_2/a_B] \\ B'' & = ([a_B/D_1 \wedge D_2])\,[a_B/a_B] \end{cases}$$

Since a_b does not occur in $D_1 \wedge D_2$.

We can now do

$$\tilde{v}_{a_B}(a_B) = v_{a_B}(a_B) = \tilde{v}(D_1 \wedge D_2) = \tilde{v}_{a_B}(D_1 \wedge D_2)$$

Apply to first and last the substitution Lemma:

$$\tilde{v}_{a_B}(B) = \tilde{v}_{a_B}(B'') \quad (\dagger\dagger)$$

Now

$$\begin{aligned} 1 &= \tilde{v}(A_i) \\ &= \tilde{v}_{a_B}(A_i) \\ &= \tilde{v}_{a_B}\left(A\left[{}^{B}/_{q}\right]\right) \\ &= \tilde{v}_{a_B}\left(A\left[{}^{B''}/_{q}\right]\right) \\ &= \tilde{v}_{a_B}\left(A\left[{}^{B'}/_{q}\right]\right) \\ &= \tilde{v}_{a_B}(A_{i+1}) \end{aligned}$$

And each equivalence, in order, is:

- by current assumption (of smth, idk)
- a_B doesn't occur in A_i
- by definition of A_i
- by substitution lemma and (††)
- by substitution lemma and (†)
- by definition of A_{i+1}

So $\tilde{v}_{a_B}(A_{i+1}) = 1$, and thus $A_{i+1} \in SAT$.

Assume now, **for the other way around**, that $A_{i+1} \in SAT$ with the aim of proving that A_i is SAT too.

Two cases:

- **First, lucky, case:** Assume further that A_{i+1} is satisfied by an assignment $w : \mathcal{L} \to \{0, 1\}$ of the form $\tilde{v}_{a_B}$, that is
$$w(a_b) = \tilde{w}(D_1 \wedge D_2)$$
at least one satisfying assignment has this property. Then we *kinda* reverse the earlier chain:
$$\begin{aligned} 1 &= \tilde{w}(A_{i+1}) \\ &= \tilde{w}\left(\left[{}^{B'}/_{q}\right]\right) \\ &= \tilde{w}\left(\left[{}^{B''}/_{q}\right]\right) \\ &= \tilde{w}\left(\left[{}^{B}/_{q}\right]\right) \\ &= \tilde{w}(A_i) \end{aligned}$$
So $\tilde{(}A_i) = 1$, that is $A_i \in SAT$.

- **Second, unlucky, case:** A_{i+1} is satisfiable but all satisfying assignments $w : \mathcal{L} \to \{0, 1\}$ ($\tilde{w}(A_{i+1}) = 1$) are such that $\tilde{w}(a_b) \neq \tilde{w}(D_1 \wedge D_2)$.
The condition
$$\tilde{w}(a_b) \neq \tilde{w}(D_1 \wedge D_2)$$
implies that
$$w(a_B) = 0 \text{ and } \tilde{w}(D_1 \wedge D_2) = 1$$
or
$$w(a_B) = 1 \text{ and } \tilde{w}(D_1 \wedge D_2) = 0$$
By bivalence, but the second one can be ruled out since $\tilde{w}$ satisfies the tail, which means that $w(a_B \to (D_1 \wedge D_2)) = 1$.

Remark 4.4.1. This technique ro reduce $SAT \preceq_p CNFSAT$ is inspired to Karp's technique to prove that

$$SAT \preceq_p 3 - CNFSAT$$

Where by $k - CNFSAT$ we mean the problem of deciding satisfiability of CNFs where each clause has exactly/at most k literals each.

Karp proved that $3 - CNFSAT$ is already $\in \mathcal{NP}c$. He also proved that $2 - CNFSAT \in \mathcal{P}$. These are called **dichotomy results**, up to a certain degree the problem is "easy", then it becomes difficult.

We want to deal with **problems of the kind** SAT, TAUTO or $\Gamma \models^? A$.
About these problems we know that:

- $SAT \preceq_p CNFSAT \in \mathcal{NP}c$
- $TAUTO \preceq_p UNSAT \preceq_p CNFUNSAT \in co - \mathcal{NP}c$

Given the problem of **deciding** whether $\Gamma \models A$ (for Γ a finite theory)

$$\Gamma = \{\gamma_1, \dots, \gamma_n\} \quad \gamma_i \in \mathcal{F}_\mathcal{L}, \quad A \in \mathcal{F}_\mathcal{L}$$

We can **reduce it** to a problem of CNFUNSAT:

$$\begin{aligned} \Gamma \models A &\Leftrightarrow \Gamma \cup \{\neg A\} \quad UNSAT(\text{ as a theory}) \\ &\Leftrightarrow \gamma_1 \wedge \gamma_2 \wedge \dots \wedge \gamma_u \wedge \neg A \quad UNSAT(\text{ as a formula}) \\ &\Leftrightarrow S \in CNF, \;\; S \in CNFUNSAT \end{aligned}$$

For some S equisat with $\gamma_1 \wedge \gamma_2 \wedge \cdots \wedge \gamma_n \wedge \neg A$.

4.5 Notational Variant for CNFs

Take a **family of formulas**

$$F_i \in \mathcal{F}_{\mathcal{L}} \quad i = 1, \ldots, k$$

Then we can "*forget*" to parenthesize these expressions due to **associativity**

$$\begin{cases} F_1 \wedge F_2 \wedge \ldots \wedge F_k \\ F_1 \vee F_2 \vee \ldots \vee F_k \end{cases}$$

We can also forget about order, due to **commutativity**

$$\begin{cases} F_1 \wedge F_2 \equiv F_2 \wedge F_1 \\ F_1 \vee F_2 \equiv F_2 \vee F_1 \end{cases}$$

And repetitions, for **idempotence**:

$$\begin{cases} F_1 \wedge F_1 \equiv F_1 \\ F_1 \vee F_1 \equiv F_1 \end{cases}$$

So from now on a **clause** $l_1 \vee l_2 \vee \ldots \vee l_u$ will be denoted as the **set of its literals** $\{l_1, l_2, \ldots, l_u\}$.
A **CNF** $C_1 \vee C_2 \vee \ldots \vee C_u$ will be denoted by the **set of its clauses** $\{C_1, C_2, \ldots, C_u\}$.

A CNF will be a set of sets of literals. A theory made of CNFs will be a set of sets of literals.

What? It says the same thing? Also on the lecture notes? TOASK

Example:

$$\begin{aligned} CNF: &\quad (p \vee q \vee \neg r) \wedge (q \vee r \vee a) \wedge p \\ &\downarrow \\ CNF: &\quad \{\{p, q, \neg r\}, \{q, r, a\}, \{p\}\} \end{aligned}$$

A theory of CNFs is just a set of clauses. An infinite theory is a set containing infinitely many clauses.

Now the notation of empty CNF as $\emptyset$ *makes sense* (thinking about an empty set of clauses), while an empty clause is $\square$ (empty set of literals).

A CNF containing an **empty clause** is **unsatisfiable**.
The shortest CNF containing $\square$ is just $\{\square\}$.
Also the empty CNF $\emptyset$ is SAT, even tautological.

We want to deal with $\Gamma \models^? A$. We want to reduce it to the problem of deciding whether a **set of clauses** S (possibly infinite) is **unsatisfiable**.

S is SAT **iff** there is an assignment $v : \mathcal{L} \to \{0, 1\}$ such that $v \models C$ for each $C \in S$ (satisfies each clause in the set). For all $C \in S$ there is at least a literal $L \in C$ such that $v \models L$ ($\tilde{v}(L) = 1$, it satisfies the literal).

If we want to prove that S is $UNSAT$, a good strategy is to **enlarge** S into a new set S' ($S \subseteq S'$) in such a way that $S \equiv S'$ (or at least S equisat S').

For instance, if enlarging S, iteratively

$$S \subseteq S' \subseteq S'' \ldots \subseteq S^{(k)}$$

if we find that $\square \in S^{(k)}$, then we can conclude that $S^{(k)}$ is UNSAT, so S is UNSAT too.

This page is intentionally left blank.

※ Refutational Methods

They are **deductive methods** whose aim is to **prove the unsatisfiability of a set of clauses** (of a formula or of a theory).

Many of these methods are based on the inference rule called "*resolution principle*".

They are calculi particularly **fit to be automated** (for proof search, we can just give theory and formula to a machine and make it decide). This is the basic behind Automated Deduction.

There are **different possible branches**:

- SAT Solver: decide if a given set of clauses is satisfiable or not.
- Theorem Prover: designed to automatically prove mathematical theorem.
- Logic Programming: asking a machine to refute a theorem $\in CNF$ (Prolog).

5.1 Resolution Principle

Given two clauses C_1 and C_2 we say that the clause D is the **resolvent** of C_1 and C_2 on the pivot L (where L is a literal) **iff**:

- $L \in C_1$
- $\overline{L} \in C_2$
- $D := (C_1 \setminus \{L\}) \cup (C_2 \setminus \{\overline{L}\})$

We shall write $D = \mathbb{R}(C_1, C_2; L, \overline{L})$.

Examples

$$S := \{\{x, y, \neg t\}, \{u, \neg y, t\}\}$$

How many resolvents can we have? 2, namely:

$$D_1 : \mathbb{R}(C_1, C_2; y, \neg y) = \{x, \neg t, u, t\}$$

$$D_2 : \mathbb{R}(C_1, C_2; \neg t, t) = \{x, y, u, \neg y\}$$

Lemma 5.1.1. Correctness (of the resolution principle, also called "Soundness").
Let $D = \mathbb{R}(C_1, C_2; L, \overline{L})$ be the resolvent of some clauses, then

$$\{C_1, C_2\} \models D$$

Proof. We must show that for each assignment $v : \mathcal{L} \to \{0, 1\}$, if $v \models C_1$ and $v \models C_2$ then $v \models D$ too. Let $v : \mathcal{L} \to \{0, 1\}$ be such that $v \models C_1$ and $v \models C_2$.

Then $v \models M$ and $v \models N$ for literals $M \in C_1$ and $N \in C_2$. If it were the case that $M = L$ and $N = \overline{L}$ then $\tilde{v}(L) = 1$ and $\tilde{v}(\overline{L}) = 1$ but this is clearly a contradiction.

Then at least one in M or N is such that $M \neq L$ or $N \leq \overline{L}$. Then we assume without loss of generality that $M \neq L$ then $M \in D$ (by definition of D), so $\tilde{v}(M) = 1$ implies that $\tilde{v}(D) = 1$.

□

Corollary 5.1.2. If $\mathbb{R}(C_1, C_2; L, \overline{L}) = \square$ then $\{C_1, C_2\}$ is UNSAT.

Proof. $\{C_1, C_2\} \models \square$ by Correctness Lemma.

□

Corollary 5.1.3. Let S be a set of clauses. Let $S = S_0, \ldots, S_k$ be a sequence of sets of clauses such that:

1. For all indexes $i = 0, 1, \ldots, k-1$, S_{i+1} is obtained from S_i by adding one or more resolvents of clauses in S_i
2. $\square \in S_k$

Then S is UNSAT.

Proof. $S = S_0 \models S_1 \models S_2 \models \ldots \models S_k \ni \square$, the sequence is guaranteed by correctness Lemma, so $S \models \square$ and S is UNSAT.

□

Our aim is to show that the **refutational method** based on iterated applications of resolution principle is **correct** and **refutationally complete**.

In general, a logical calculus may have (or not) the two following **properties**:

- **Correctness (Soundness):** a calculus is correct (sound) if the "*certificates*" it outputs (the proofs) witness a true state of things (it does not produce "fake certificates", falsifying assignment). In our case the sequence $S_0, S_1, \ldots, S_k \ni \square$ is a correct "certificate", or proof, that S is UNSAT.

- **Completeness:** a calculus is complete if it does not omit any "certificate" (proof). In our case it means that to be complete our method should have the following property: If S is UNSAT then there is a sequence (certificate) $S = S_0, \ldots, S_k \ni \square$ ($S_i \rightsquigarrow S_{i+1}$ only adding resolvents from clauses in S_i).

Our refutation, resolution-based, method we'll only be **refutationally complete**.

5.1.1 Refutational Completeness of the Resolution Principle (proof of J.A. Robinson)

A (possibly infinite) set of clauses S is unsatisfiable iff (since we're proving correctness and completeness) $\square \in \mathcal{R}^*(S)$ where $\mathcal{R}^*(S)$ is defined as follows:

$$\begin{aligned}
\mathcal{R}(S) &:= S \cup \{D : D = \mathbb{R}(C_1, C_2; L, \overline{L}) \text{ for some } C_1, C_2 \in S, L \in C_1, \overline{L} \in C_2\} \\
\mathcal{R}^2(S) &:= \mathcal{R}\left(\mathcal{R}(S)\right) \\
&\vdots \\
\mathcal{R}^{t+1}(S) &:= \mathcal{R}\left(\mathcal{R}^t(S)\right) \\
&\vdots \\
\mathcal{R}^*(S) &:= \textstyle\bigcup_{i \in \omega} \mathcal{R}^i(S)
\end{aligned}$$

For simplicity

$$\mathcal{R}^1(S) := \mathcal{R}(S)$$
$$\mathcal{R}^0(S) := S$$

Remark 5.1.4. Assume we proved Robinson's theorem. If S is a finite set of clauses, then $\mathcal{R}^*$ provides us with a decision procedure for establishing whether S is SAT or UNSAT. That is, in both cases, the procedure building $\mathcal{R}^*(S)$ ends in a finite number of steps (finite amount of time) providing always the correct answer.

Proof. If S is finite then in S there occur only finitely many distinct propositional letters. Let us write $Var(S) \subseteq \{p_1, p_2, \ldots, p_n\}$ for some $n \in \mathbb{N}$.

The literals writable using only $p_1, p_2, \ldots, p_n$ are exactly 2^n.
The clauses writable using only literals on $p_1, p_2, \ldots, p_n$ are exactly 2^{2n}.

Key observation: the application of the resolution rule (the **formation of new resolvents**) **never introduces any new literals**.

So, the sequence

$$(\dagger) \quad S = \mathcal{R}^0(S) \subseteq \mathcal{R}^1(S) \subseteq \ \ldots \ \subseteq \mathcal{R}^k(S) \subseteq \ \ldots$$

is such that every $\mathcal{R}^i(S)$ is a subset of the 2^{2n} clauses writable on $p_1, p_2, \ \ldots \ , p_n$.

Let us write $\mathcal{C}^{(n)}$ for the set of all clauses on $p_1, \ \ldots \ , p_n$

$$\mathcal{R}^i(S) \subseteq \mathcal{C}^{(n)} \ \text{ for each } \ i \in \omega$$

Then there must be $t \in \omega$ such that $\mathcal{R}^t(S) = \mathcal{R}^{t+1}(S)$ (since no $\mathcal{R}^i(S)$ can have more that 2^{2n} clauses, $t \leq 2^{2n}$).

Then

$$\mathcal{R}^t(S) = \mathcal{R}^{t+1}(S) = \mathcal{R}\left(\mathcal{R}^t(S)\right) = \mathcal{R}\left(\mathcal{R}^{t+1}(S)\right) = \mathcal{R}^{t+2}(S) = \ \ldots$$

So we have that

$$\mathcal{R}^t(S) = \mathcal{R}^{t+1}(S) = \ \ldots \ = \mathcal{R}^*(S)$$

Then: At step $i \rightsquigarrow i+1$ of the construction:

- if we found $\square \in \mathcal{R}^{i+1}(S)$ then stop $\implies$ output S UNSAT
- If $\square \notin \mathcal{R}^{i+1}(S)$ then
  - if $\mathcal{R}^{i+1}(S) = \mathcal{R}^i(S)$ then stop and output S SAT
  - else go to the next step

This algorithm **always terminates when S is finite**.

□

Note: If S is **infinite**, but it happens that $\mathcal{R}^t(S) = \mathcal{R}^{t+1}(S)$ (and we can check this), the algorithm can stop and say that S is SAT. The procedure is complete, but it's a semi-decision procedure, might not end.

Correctness

Assuming that $\square \in \mathcal{R}^*(S)$, then, by definition of $\mathcal{R}^*(S) = \bigcup_{i \in \omega} \mathcal{R}^i(S)$, there is $i \in \omega$ such that $\square \in \mathcal{R}^i(S)$.

So $\mathcal{R}^i(S)$ is UNSAT, by definition of $\square$.
But $\mathcal{R}^i(S) \equiv \mathcal{R}^{i-1}(S) \equiv \ \ldots \ \equiv \mathcal{R}^0(S) \equiv S$ (by Correctness Lemma). So S is UNSAT.

(Refutational) Completeness

We have to prove that S UNSAT implies $\square \in \mathcal{R}^*(S)$.

Proof. Since $S \in UNSAT$ (by assumption), by compactness theorem, there is a $S_{fin} \subseteq_\omega S$ such that S_{fin} is UNSAT.

In S_{fin} there occur finitely many (distinct) propositional letters, say $Var(S_{fin}) \subseteq \{p_1, \ldots, p_n\} \subseteq \mathcal{L}$, for some $n \in \omega$.

Remark 5.1.5. $S_{fin} \subseteq \mathcal{C}^{(n)}$ (we call $\mathcal{C}^{(n)}$ the set of all clauses over the letters $p_1, \ldots, p_n$).

Remark 5.1.6. $\mathcal{C}^{(0)} = \{\square\}$ ($|\mathcal{C}^{(0)}| = 2^{2 \cdot 0} = 1$).

A fortiori ("from stronger reasons"): we can say that

$$\left.\begin{array}{ll} S_{fin} \subseteq & \mathcal{C}^{(n)} \\ S_{fin} \subseteq & S \end{array}\right\} \Longrightarrow S_{fin} \subseteq \mathcal{C}^{(n)} \cap \mathcal{R}^*(S)$$

(since $S \subseteq \mathcal{R}^*(S)$).

It follows

$$(\dagger\dagger) \qquad \mathcal{C}^{(n)} \cap \mathfrak{R}^*(S) \text{ is } UNSAT$$

Since it contains S_{fin}, which is UNSAT by definition.

We shall prove that for each $k = n, n-1, \ldots, 0$ that $\mathcal{C}^{(k)} \cap \mathcal{R}^*(S)$ is UNSAT ($\dagger$).

If we can prove ($\dagger$) for all k, then ($\dagger$) holds for $k = 0$ too. That is $\mathcal{C}^{(0)} \cap \mathcal{R}^*(S)$ is UNSAT, and thus $\{\square\} \cap \mathcal{R}^*(S)$ is UNSAT.

If we prove $\{\square\} \cap \mathcal{R}^*(S)$ is UNSAT ($\dagger\dagger\dagger$) then

$$\{\square\} \cap \mathcal{R}^*(S) = \begin{cases} \emptyset & \rightarrow \{\square\} \cap \mathcal{R}^*(S) \; SAT \text{ contradicting } (\dagger\dagger\dagger) \\ \text{or} & \\ \{\square\} & \rightarrow \{\square\} \cap \mathcal{R}^*(S) \; UNSAT \end{cases}$$

but

$$\{\square\} \cap \mathcal{R}^*(S) = \{\square\} \text{ iff } \square \in \mathcal{R}^*(S)$$

And this is the end of completeness.

□

We shall prove by **descending induction** on the index $k = n, n-1, \ldots, 0$ that (†): $\mathcal{C}^{(k)} \cap \mathcal{R}^*(S)$ is UNSAT.

Base: $k = n$, then $\mathcal{C}^{(k)} \cap \mathcal{R}$ UNSAT, which is already proven, by (††).

Step: Assume the statement (†) is true for $k = n, k = n-1, 1, \ldots, k+1$ and we want to prove it for k.
So $\mathcal{C}^{(n)} \cap \mathcal{R}^*(S)$ UNSAT, $\mathcal{C}^{(n-1)} \cap \mathcal{R}^*(S)$ UNSAT, ...$\mathcal{C}^{(k+1)} \cap \mathcal{R}^*(S)$ UNSAT, and we want to **prove** $\mathcal{C}^{(k)} \cap \mathcal{R}^*(S)$ UNSAT, too.

By **contradiction**: we assume $\mathcal{C}^{(k)} \cap \mathcal{R}^*(S)$ is SAT.
Then there is $v : \mathcal{L} \to \{0, 1\}$ such that for all $C \in \mathcal{C}^{(k)} \cap \mathcal{R}^*(S)$, $v \models C$ $(\tilde{v}(C) = 1)$.

Let us define the two following variants $v^+, v^- : \mathcal{L} \to \{0, 1\}$ of v:

- $v^+(p_{k+1}) := 1$
- $v^-(p_{k+1}) := 0$
- $v^+(p_i) := v(p_i) =: v^-(p_i)$ for all $i \neq k+1$

By I.H. $\mathcal{C}^{(k+1)} \cap \mathcal{R}^*(S)$ is UNSAT, then $v, v^+, v^- \not\models \mathcal{C}^{(k+1)} \cap \mathcal{R}^*(S)$.

Then there is at least a clause $C_1 \in \mathcal{C}^{(k+1)} \cap \mathcal{R}^*(S)$ such that $\tilde{v}^+(C_1) = 0$ $(v^+ \not\models C_1)$ $(\star)$, analogously there must be a $C_2 \in \mathcal{C}^{(k+1)} \cap \mathcal{R}^*(S)$ such that $\tilde{v}^-(C_2) = 0$ $(v^- \not\models C_2)$ $(\star\star)$.

Key Observation: p_{k+1}, as a propositional letter, occurs in C_1 and, precisely, only in the literal $\neg p_{k+1}$ (same thing but not negated in C_2)

$$\neg p_{k+1} \in C_1, \quad p_{k+1} \notin C_1$$

Proof. As a matter of fact

- p_{k+1} cannot occur in C_1 as a literal, for otherwise $v^+(p_{k+1}) = 1$ implies $\tilde{v}^+(C_1) = 1$, contradicting $(\star)$
- p_{k+1} must occur in C_1 in the literal $\neg p_{k+1}$, for otherwise $p_{k+1} \notin C_1$, $\neg p_{k+1} \notin C_1$, so $C_1 \in \mathcal{C}^{(k)} \cap \mathcal{R}^*(S)$, and so $\tilde{v}(C_1) = 1$, but since p_{k+1} does not occur as a variable in C_1, $\tilde{v}^+(C_1) = \tilde{v}(C_1) = 1$ that is $\tilde{v}^+(C_1) = 1$, contradicting $(\star)$.

So $\neg p_{k+1} \in C_1$, $p_{k+1} \notin C_1$.

□

Analogously: p_{k+1} must be in C_2 in the form p_{k+1}.

Then $\neg p_{k+1} \in C_1$, $p_{k+1} \notin C_1$, $p_{k+1} \in C_2$, $\neg p_{k+1} \notin C_2$.

Then there **exists the resolvent**

$$\begin{aligned} D &= \mathbb{R}(C_1, C_2; \neg p_{k+1}, p_{k+1}) \\ &= (C_1 \setminus \{\neg p_{k+1}\}) \cup (C_2 \setminus \{p_{k+1}\}) \\ &= (C_1 \cup C_2) \setminus \{p_{k+1}, \neg p_{k+1}\} \end{aligned}$$

Notice that D is such that $\neg p_{k+1} \notin D$ and $p_{k+1} \notin D$, so

$$D \in \mathcal{C}^{(k)} \cap \mathcal{R}^*(S) \implies \tilde{v}(D) = 1$$

Then v must satisfy at least one literal in $m \in D$, $(\tilde{v}(M) = 1)$.

So $M \in D = (C_1 \cup C_2) \setminus \{p_{k+1}, \neg p_{k+1}\}$, and $\tilde{v}(M) = 1$, so we have two cases:

1. $M \in C_1 \setminus \{\neg p_{k+1}, p_{k+1}\}$, then $\tilde{v}(C_1) = 1$, and then since p_{k+1} does not occur in C_1 (as a letter), $\tilde{v}^+(C_1) = \tilde{v}(C_1) = 1$, contradicting $(\star)$.

2. $M \in C_2 \setminus \{p_{k+1}, \neg p_{k+1}\}$, then $\tilde{v}(C_2) = 1$, and then since p_{k+1} does not occur in C_2 (as a letter), $\tilde{v}^-(C_2) = \tilde{v}(C_2) = 1$, contradicting $(\star\star)$.

As there are no more cases to consider we have concluded our proof by contradiction, showing that it is not the case that $\mathcal{C}^{(k)} \cap \mathcal{R}^*(S)$ is SAT, implying, in any case, that $\mathcal{C}^{(k)} \cap \mathcal{R}^*(S)$ is UNSAT.

This concludes induction step and proof by induction. We have proved that $\mathcal{C}^{(k)} \cap \mathcal{R}^*(S)$ is UNSAT for all $k = n, \ldots, 0$.

Remark 5.1.7. Say S is an **infinite theory**. In the first step of the proof we reduced S to $S_{fin} \subseteq_\omega S$ by compactness theorem.

This application of compactness theorem doesn't provide us with a decision procedure for the input S, when S is infinite, since, in general, we don't know anything useful on the concrete nature of S_{fin} (we have no clues on how to choose such $S_{fin} \subseteq_\omega S$, S_{fin} UNSAT).

The best thing we can do is to **build a possibly infinite sequence of finite subsets** of S:

$$S_1 \subseteq_\omega S_2 \subseteq_\omega \ldots \subseteq_\omega S \text{ s. th. } \bigcup_{i \in \omega} S_i = S$$

Then, for each S_i, which are all finite, compute $\mathcal{R}^*(S_i)$.

Remark 5.1.8. If S is infinite, at step i:

- If $\square \in \mathcal{R}^m(S_i)$ for some $m \in \omega$:

$$\begin{array}{rl} \Longrightarrow & \square \in \mathcal{R}^*(S_i) \\ \Longrightarrow & \square \in \mathcal{R}^*(S) \\ \Longrightarrow & S \;\; UNSAT \end{array}$$

- If we find $t \in \omega$, $\square \notin \mathcal{R}^t(S_i)$ and $\mathcal{R}^t(S_i) = \mathcal{R}^{t+1}(S_i)$, then S_i is SAT, go to the next step $i+1$

If S_i UNSAT (which is decidable) then S UNSAT and stop, but if S_i is SAT (also decidable) then go on. This is a semi-decidability procedure for "S SAT?".

Definition 5.1.9. A **deduction by resolution** of a clause C from a set of clauses S ($S \vdash_R C$) is a finite sequence $C_1, C_2, \ldots, C_u$ of clauses such that

1. $C_u = C$
2. For all indexes $i \in \{1, \ldots, u\}$ either
   (a) $C_i \in S$
   (b) $C_i = \mathbb{R}(C_j, C_k; L, \overline{L})$ for some $j, k < i$ (and some literals $L \in C_j$, $\overline{L} \in C_k$)

$S \vdash_R C$ by correctness, implies that $S \models C$.

A **deduction by resolution of** $\square$ from S ($S \vdash_R \square$) is called a **refutation** of S (and then S is UNSAT, i.e. $S \models \square$).

The **refutational completeness** of $\vdash_R$ **shows**

$$\begin{array}{rl} & S \text{ UNSAT iff } S \vdash_R \square \\ \text{equivalently} & S \models \square \text{ iff } S \vdash_R \square \\ \text{equivalently} & S \models \bot \text{ iff } S \vdash_R \bot^c \end{array}$$

A calculus C is **refutationally complete iff** $\Gamma \models \bot$ implies $\Gamma \vdash_C \bot$ (whenever Γ is contradictory, i.e., is UNSAT, then the calculus C proves such contradiction).

A calculus C is **"tout-court"/fully complete** (just complete) **iff** $\Gamma \models A$ implies $\Gamma \vdash_C A$ for any $A \in \mathcal{F}_{\mathcal{L}}$.

Examples of deductions (not necessarily refutations) in $\vdash_R$

$$a, b, c \in \mathcal{L} \quad S = \{\{a, b, \neg c\}, \{a, b, c\}, \{a, \neg b\}\} \models^? \{\{a\}\}$$

Steps:

0. Transform $S \models^? \{a\}$ into $S \cup \{\neg a\}$ UNSAT? That is, show that $S, \neg a \vdash_R \Box$
1. Choose a formula from S, $\{a, b, \neg c\} \in S$
2. Choose another formula? $\{a, b, c\}$
3. Choose $\{a, \neg b\}$
4. Choose $\{\neg a\}$
5. Resolve 1 and 2 together: $\mathbb{R}(1, 2)$: $\{a, b\}$
6. Resolve 3 and 5 together: $\mathbb{R}(3, 5)$: $\{a\}$
7. Resolve 4 and 6 together: $\mathbb{R}(4, 6)$: $\Box$

The first 6 steps constitute a direct deduction from $S, \neg a \vdash_R \{a\}$, a deduction by resolution of A from S.

Consider a theory $\Gamma \subseteq \mathcal{F}_\mathcal{L}$, a formula $A \in \mathcal{F}_\mathcal{L}$.
Let Γ^C be the set of clauses equisatisfiable with Γ. Let A^C ($(\neg A)^C$) be a set of clauses equisat with A ($\neg A$).

$$\Gamma \models A \Leftrightarrow \Gamma, \neg A \ \ UNSAT \Leftrightarrow \Gamma^c, (\neg A)^C \vdash_R \Box \Leftarrow \Gamma^C \vdash_R A^C$$

full completeness fails for $\vdash_R$.

(†)

So (†) shows that $\vdash_R$ is only refutationally complete, that is

$$\Gamma \models \bot \Leftrightarrow \Gamma \vdash_C \bot$$

$$(S \models \Box \Leftrightarrow S \vdash_R \Box)$$

but it's not complete (so

This is not a problem from the computational point of view, because, via a suitable and efficient "pre-processing", we reduce $\Gamma \models^? A$ to a refutational problem $\Gamma^C, (\neg A)^C \vdash_R^? \Box$.

Actually, from the computational point of view, calculi that are just refutationally complete, may be preferable for efficiency reasons with respect to fully complete calculi.

5.2 Axiomatic Systems (Hilbert's style calculus)

The Hilbert's style calculus $\mathcal{H}$ is another formal logical system that can be used for mathematical logic. It's **based on a set of axioms** and inference rules that are designed to formalize mathematical reasoning.

It's relatively short but **not well suited to automated deduction**, due to it's nature.

The **axioms** are:

- $A_1 : A \to (B \to A)$
- $A_2 : (A \to (B \to C)) \to ((A \to B) \to (A \to C))$
- $A_3 : (\neg B \to \neg A) \to (A \to B)$

Inference rule(s):

- Modus Ponens:

$$\frac{A \qquad A \to B}{B}$$

A **proof** of $A \in \mathcal{F}_\mathcal{L}$, from some theory $\Gamma \subseteq \mathcal{F}_\mathcal{L}$ in the **Hilbert Style calculus** $\Gamma \vdash_\mathcal{H} A$ is a **finite sequence** $A_1, \ldots, A_u$ **of formulas such that**:

1. $A_u = A$
2. For each $i \in \{1, 2, \ldots, u\}$ it holds that either
   - $A_i \in \Gamma$, or
   - A_i is an instance of one of the axiom schemes above, or
   - there are $j, k < i$ such that $A_j = A_k \to A_i$ so that

We have proof that $\mathcal{H}$ is **sound** and **complete** (fully, not just refutationally)

$$\Gamma \models A \text{ iff } \Gamma \vdash_\mathcal{H} A$$

As an **example** we will verify that

$$\neg\neg A \models^{?} A$$

First we will try using **resolution**. First of all we substitute the formula A for a propositional letter, obtaining

$$\neg\neg p \overset{?}{\models} p$$

We then change the problem in an unsatisfiability one

$$\{\neg\neg p, \neg p\} \quad UNSAT$$

finally we normalize

$$\{\{p\}, \{\neg p\}\} \vdash_R \square$$

which has proof

$$\frac{p \qquad \neg p}{\square}$$

In $\mathcal{H}$ we do not have any preprocessing to do. Steps:

$$\dfrac{\dfrac{\dfrac{Ax_1\ \dfrac{}{\neg\neg A \to (\neg\neg\neg\neg A \to \neg\neg A)}}{\text{MP}\ \dfrac{\neg\neg\neg\neg A \to \neg\neg A}{}} \qquad \neg\neg A}{\text{MP}\ \dfrac{\neg\neg A \to A}{}} \qquad \dfrac{Ax_3\ \dfrac{}{(\neg\neg\neg\neg A \to \neg\neg A) \to (\neg A \to \neg\neg\neg A)}}{} \qquad \neg\neg A}{A}$$

5.3 Refutational procedure by David Putnam (DPP)

Problem: Maintaining (refutational) completeness by selecting all possible resolvents.

Remark 5.3.1. Definition: A clause C_1 **subsumes** C_2 (or C_2 is subsumed by C_1) **iff** $C_1 \subset C_2$ (Notice: $\models C_1 \implies C_2$, it's an implication and a tautology, equivalently $\{C_1, C_2\} \equiv \{C_1\}$).

Subsumption rule: Let S be a set of clauses and let S' be obtained from S, by removing all subsumed clauses. Then $S \equiv S'$, preserving logical equivalence. Notice S' cannot be further reduced by subsumption. One subset of the other, elminate the bigger one.

Remark 5.3.2. Definition: A clause C is **trivial iff** it contains L and $\overline{L}$ for some literal L. Notice $\models C$, if $C \in S$ then $S \equiv S \setminus \{C\}$.

Elimination of trivial clauses rule: Let S be a set of clauses and let S' be obtained from S by removing all trivial clauses. Then $S \equiv S'$. Remove trivial clauses, they don't matter.

Notice: If $D = \mathbb{R}(C_1, C_2; L, \overline{L})$ and C_1, C_2 are not trivial

$$(C_1 \setminus \{L\}) \cup (C_2 \setminus \{\overline{L}\}) = (C_1 \cup C_2) \setminus \{L, \overline{L}\}$$

While if it was trivial

$$\begin{aligned}\mathbb{R}(\{a, \neg a\}, \{a, \neg a\}, a, \neg a) &= (\{a, \neg a\} \setminus \{a\}) \cup (\{a, \neg a\} \setminus \{\neg a\}) \\ &= \{\neg a\} \cup \{a\} = \{a, \neg a\} \\ &\neq (\{a, \neg a\} \cup \{a, \neg a\}) \setminus \{a, \neg a\} = \square\end{aligned}$$

So, **trivial clauses do not carry meaningful information** and removing them allows us to use resolvents in the easier form.

5.3.1 Steps

Input for a step of DPP: A **finite set of clauses** S, S is already "**cleaned**" from **subsumed and trivial clauses** (we have applied to the original input both the subsumption rule and the elimination of trivial clauses rule).

We call the process of "**cleaning**"

$$S = clean(S)$$

Then each step of the DPP **consists of**:

1. **Choose** a **propositional letter** $p \in \mathcal{L}$ among those occurring in clauses of S. p is called the "**pivot**" the step. How do we choose? No fixed way, heuristically.

2. **Form**
$$S' = \{\mathcal{C} \in S : p \notin \mathcal{C} \text{ and } \neg p \notin C\}$$
S' is the set of p**-free clauses** of S.

3. **Form**
$$S'' = \{D = \mathbb{R}(C_1, C_2; p, \neg p) : C_1, C_2 \in S \setminus S'\}$$
S'' is the set of p**-resolvents** of S.

4. **Form**
$$S''' = clean(S' \cup S'')$$
The cleaned version of S''.

5. **Output** S''' and go to the **next step**, i.e., repeat the process with another literal.

If we get $\square$ then S UNSAT, otherwise we get $\emptyset$ and S is SAT.

Example 5.3.3. Let

$$S = \{\{a, b, \neg c\}, \{a, \neg b, d\}, \{a, \neg b, \neg c, \neg c\}, \{\neg a, d\}, \{\neg a, \neg c, \neg d\}, \{c\}\}$$

- we choose a pivot, an heuristic is choosing a literal appearing in the shortest clause. In this case c;
- we create the c-free set of clauses

$$\{\{a, \neg b, d\}, \{\neg a, d\}\}$$

- we create the c-resolvents

$$\{\{a, b\}, \{a, \neg b, \neg d\}, \{\neg a, \neg d\}\}$$

- we clean the set, obtaining

$$\{\{a, \neg b, d\}, \{\neg a, d\}, \{a, b\}, \{a, \neg b, \neg d\}, \{\neg a, \neg d\}\}$$

  because there was nothing to clean;
- we choose a new pivot: a;
- we form the a-free set of clauses

$$\varnothing$$

- we form the a-resolvent se

$$\{\{\neg b, d\}, \{b, d\}, \{\neg b, d, \neg d\}, \{b, \neg d\}, \{\neg b, \neg d\}\}$$

- we clean the set, obtaining

$$\{\{\neg b, d\}, \{b, d\}, \{b, \neg d\}, \{\neg b, \neg d\}\}$$

- we choose a new pivot: b;
- we form the b-free set of clauses

$$\varnothing$$

- we form the b-resolvents:

$$\{\{d\}, \{d, \neg d\}, \{\neg d\}\}$$

- we clean the set

$$\{\{d\}, \{\neg d\}\}$$

- we choose a new pivot: d;
- we form the d-free set

$$\varnothing$$

- we form the d-resolvents

$$\{\square\}$$

  S is unsatisfiable.

Example 5.3.4. Let

$$S = \{\{a, \neg b, c\}, \{b, \neg c\}, \{a, c, \neg d\}, \{\neg b, \neg d\}, \{a, b, d\}, \{\neg a, d, b\}, \{b, \neg c, d\}, \{c, \neg c\}\}$$

this is cleaned into

$$S = \{\{a, \neg b, c\}, \{b, \neg c\}, \{a, c, \neg d\}, \{\neg b, \neg d\}, \{a, b, d\}, \{\neg a, d, b\}\}$$

- we pivot on b;
- we make the b-free set
$$\{\{a, c, \neg d\}\}$$
- we make the b-resolvents
$$\{\{a, c, \neg c\}, \{a, c, d\}, \{a, \neg a, c, d\}, \{\neg c, \neg d\}, \{a, d, \neg d\}, \{\neg a, d, \neg d\}\}$$
- we clean the set
$$\{\{a, c, \neg d\}, \{a, c, d\}, \{\neg c, \neg d\}\}$$
- we pivot on c;
- we make the c-free set
$$\varnothing$$
- we make the c-resolvents
$$\{\{a, \neg d\}, \{a, d, \neg d\}\}$$
- we clean the set
$$\{\{a, \neg d\}\}$$
- we pivot on a;
- we make the a-free set
$$\varnothing$$
- we make the a-resolvents
$$\varnothing$$
- we clean, obtaining
$$\varnothing$$
- S is satisfiable.

5.3.2 Model Building

Referring to the last example. We shall use the DPP proof to get a **satisfying assignment** for S.

If a **letter** has **never been used as a pivot** in any step let us call it "**outsider**". To the outsiders we assign **whatever truth value**. So, in our earlier example $v(d) = 0$.

Let us **substitute** the **values already assigned to each step**, proceeding **backwards from the last step** $\neq \emptyset$. So we start with:

$$S_2 = \{\{a, \neg d\}\}$$

With $v(d)$ known, define $v(a)$ in order to satisfy S_2: $v(a) = 0$.

Going backwards:

$$S_1 = \{\{a, c, d\}, \{\neg c, \neg d\}, \{a, c, \neg d\}\}$$

Pivot of $S_1 \rightsquigarrow S_2$ is c. We now find a suitable $v(c)$ and looking at the first clause we can say that $v(c) = 1$

Last step:

$$S_0 = \{\{a, \neg b, c\}, \{b, \neg c\}, \{a, c, \neg d\}, \{\neg b, \neg d\}, \{a, b, d\}, \{\neg a, b, d\}\}$$

The pivot was b. We need to chose $v(b)$, looking at the already satisfied ones we see that the choice must be $v(b) = 1$.

5.3.3 Termination

Proof. Let us display the DPP over input S as the sequence of the outputs of the steps $DPP(S) = S_0, S_1, \ldots, S_k$ with $S_0 = clean(S)$.

S_i is obtained from S_{i-1} by one DPP step on pivot q_i.

Observe that S_i does not contain any occurrence of q_i (as a propositional letter).

After at most $k \leq n$ steps it holds that $Var(S_k) = \emptyset$ (there are no more letters).

So, either $S_k = \{\square\}$ or $S_k = \emptyset$.

□

We're removing letters at each step, after some time it must end.

5.3.4 Correctness and refutational completeness of DPP

Theorem 5.3.5. Correctness and (refutational) completeness of the DPP.
A finite set of clauses S in the propositional letters $p_1, p_2, \ldots, p_n$ is **UNSAT iff** within at most n DPP steps $\square$ **is produced**.

Alternatively, S is **SAT iff** within at most n DPP steps $\emptyset$ **is produced**.

This is takes a **linear number of steps**, takes n steps, since we get rid of at least one letter at each step, but each step itself is not linear in time, so we, sadly, did not prove that $\mathcal{P} = \mathcal{NP}$.

We call k the last step of DPP on input S, obviously $k \leq n$ where $Var(s) \subseteq \{p_1, \ldots p_n\}$.

To prove Correctness one of two forms:

- If we reach $\square$, $S_k = \{\square\}$, then S_0 (first step, "cleaning" of S) is UNSAT
- if S_0 is SAT then $S_k = \emptyset$

To prove Refutational completeness one of two forms:

- If S_0 is UNSAT then $S_k = \{\square\}$
- if $S_k = \emptyset$ then S_0 is SAT (we'll prove this form)

Correctness

If $S_k = \{\square\}$ then S_0 is UNSAT.

Proof. It's just a **corollary of the correctness lemma**, since $S_i \models S_{i+1}$ then $S_0 \models \{\square\}$, that is S_0 is UNSAT.

$\square$

Refutational Completeness

We have to prove that $S_k = \emptyset$ implies S_0 is SAT.

Proof: By descending induction on k. We shall prove S_i SAT for $i = k, k-1, \ldots 1, 0$.

Base: $i = k$, $S_i = S_k = \emptyset$. By definition of $\emptyset$, this is SAT.

Step: Assume S_{i+1} is SAT, with the aim of proving S_i SAT, too.
Then we know that there is $v : \mathcal{L} \to \{0,1\}$ such that $v \models S_{i+1}$.
We define two variants of v, $v^-, v^+ : \mathcal{L} \to \{0,1\}$: * $v^+(q_{i+1}) = 1$ * $v^-(q_{i+1}) = 0$ $v^+(p) = v = v^-(p)$ for all other $p \in \mathcal{L}$, $p \neq p_{i+1}$.
p_{i+1} is the pivot in step S_{i+1}.

We shall show that either $v^+ \models S_i$ or $v^- \models S_i$, that is, S_i is SAT (†).

By contraditicon we asssume that $v^+ \not\models S_i$ and $v^- \not\models S_i$ (††).

Than there are $C_1, C_2 \in S_i$ such that $v^+ \not\models C_1$ and $v^- \not\models C_2$.

Key observation: $\neg q_{i+1} \in C_1$ and $q_{i+1} \notin C_1$ (note that C_1 is not trivial). For otherwise, if $q_{i+1} \in C_1$ (as a literal) then $v^+(C_1) = 1$ contradicting (††).
Then $q_{i+1} \notin C_1$ (as a literal). If $\neg q_{i+1} \notin C_1$, too, then C_1 does not contain q_{i+1}, so $C_1 \in S_{i+1}$, then $v \models C_1$, but then $v^+ \models C_1$ (since $q_{i+1} \notin Var(C_1)$), reaching another contradiction of (††).

Analogously $q_{i+1} \in C_2$ and $\neg q_{i+1} \notin C_2$ (noting that C_2 is not trivial).

Then we can form the resolvent

$$D = \mathbb{R}(C_1, C_2; \neg q_{i+1}, q_{i+1})$$

and D is not trivial since C_1 and C_2 are not.
So $D \in q_{i+1}$-resolvents $\implies$ either $D \in S_{i+1}$ or there is $D' \in S_{i+1}$ and D' subsumes D.
In both cases $v \models D$.

$v \models D$ implies $v^+ \models D$ and $(v^- \models D)$, since ...

We have reached in any case contradiction. We proved by contradiction that either $v^+ \models S_i$ or $v^- \models S_i$, that is S_i SAT.

Then S_0 is SAT.

Remarks Let S be a finite set of clauses. Then $Var(S) \subseteq \{p_1, \ldots, p_n\}$ for some $n \leq ||S||$.
Then

$$DPP(S) = S_0, S_1, \ldots, S_k$$

with $k \leq n \leq ||S||$.

If we can guarantee that $||S_i||$ is always polynomially bounded by the size of the input $||S||$ then the overall length of the DPP proof, $||S_0, \ldots, S_k||$ would be polynomially short with respect to $||S||$.

If the choice of the pivot at each step is polynomially short with respect to $||S||$ the overall time used by $DPP(S)$ would be polynomial with respect to $||S||$, too.

There are fragments of CNF on which DPP runs on polynomial time. Notable examples: * $2CNFSAT \in \mathcal{P}$ (also known as KROMSAT) * $HORNSAT \in \mathcal{P}$ (A clause is Horn iff it contains at most one positive literal)

Haken: Every refutational (i.e., based on resolution) prook of the pigeon-hole-principle on n pigeons PHP_n will require at least 2^n steps (computational steps, in the computational model, Turing Machines in our case).

PHP_n, we have: * P: pigeons * H: holes * $|H| = n$ * $|P| = n + 1$

One more pigeons than holes.

Improvements on DPP: DPLL (Davis-Putnam-Loveland-Logemann).
On input S (finite set of clauses) **tries** to build a satisfying assignment by extending a partial assignment. Split rule: at a certain point assign at the pivot either 0 or 1, fork the possibilities for eventual backtracking.
Saves space with respect to DPP, by using backtracking.

Unit propagation: * unit resoluzion: resolvents only among singletons and Clauses * unit subsumption: if $L \in C \implies$ delete C (if you're assuming $v(L) = 1$)

Improvement on DPLL: CDCL: Conflict Driven Clause Learning

DPLL + non-chronological backtracking (backjumping). Building an implication graph while building the DPLL which is able to find conflict clauses (not satisfied by the partial assignment of the DPLL).

Learn the clause: add a clause recording the conflict to the input and Backtrack to the first choice that originated the conflict

First-Order Logic

Introductory example: If it rains (p) I take the umbrella (q) I do not take the umbrella ($\neg q$) *** Therefore it does not rain

This lies in propositional logic:

$$\{p \to q, \neg q\} \models \neg p$$

Transform it into UNSAT

DPP on

Outputs $\square$, so the thingy is UNSAT, then the first thingy is correct.

Another one:

Every man is mortal (p) Socrates is a man (q) *** Therefore Socrates is a mortal (r)

They are all atomic, we can't relate them using propositional logic, trying DPP on this is straight up dumb, starting from $\{p, q\} \models r$, obviously it goes to $\emptyset$. We need a more expressive language able to express these kind of concepts.

To handle reasoning such as the syllogism about Socrates we must adopt richer (more expressive) language.

Syntax: We enrich syntax by adding new kinds of symbols: * Quantifiers * Predicate symbols * Function symbols (including constants) * Variables

Semantics: We attach new meanings to new concepts, to assign a formal way to intepret the new "ingredients" of the language, with the aim of being able to express more refined aspects of the possible "worlds" that we want to model.

Try to develop calculi fit to be automated , to (semi) decide problems like: Every man is mortal (p) Socrates is a man (q) *** Therefore Socrates is a mortal (r) –> $\forall x(U(x) \to M(x))$ $U(s)$ *** M(s)

So

$$\{\forall x(U(x) \to M(x)), U(s)\} \models^{?} M(s)$$

But it will make sense after introducing syntax and semantics.

Preprocessing, going in Normal Form, getting first-order clauses and another UNSAT problem

$$\{\{\neg U(x), M(x)\}, \{U(s)\}, \{\neg M(s)\}\} \quad UNSAT$$

Herbrand theory: we may substitute x with s

We can all cosider these propositional letters:

Then we can just use resolution, showing that it's refutable and thus proving true the first statement.

Every man is mortal The father of every man is a man Socrates is a man *** The father of Socrates is mortal

$\forall x(U(x) \to M(x))$ $\forall x(U(x) \to U(f(x)))$ $U(s)$ *** $M(f(s))$

Where the function f represents a relation (father) to an element, Socrates s in this case.

$$\{\{\neg U(x), M(x)\}, \}$$

Using herbrand's theory: we can use s and $f(s)$, but also $f(f(s))$, $f(f(f(s)))$, ..., there are infinitely many different substitutions. We need to find the "clever" one, not just any.
In this case x becomes $f(s)$