1 Introduction to Logic

1.1 History of logic

Humans are argumentative by nature, one then wants to know if a sequence of arguments makes sense, whether a deduction made from a sequence of premise is correct. The very first such recorded deduction can be attributed to Socrates. You may have heard " All men are mortal, Socrates is a man and hence also mortal". Leibniz was probably the very first to codify a calculus by which one could systematically construct and validate arguments. This was later characterised by George Bool as propositional logic.

A more expressive logic than the propositional logic is what is called predicate logic. This logic can be attributed to Frege and Pierce. The logic that we study today is result of extensive work by Russel, Hilbert, Gödel and many others. The need for it was accelerated by very many paradoxes that were discovered. Logic provided as a consistent framework to prove mathematical theorems.

Logic is fundamental to many areas of science. Computer Science is one area that has benefited the most from it. In fact, the notion of computation was born out of logic.

2 Propositional Logic

2.1 Syntax and Semantics

We will present syntax and semantics of propositional logic. We assume a countable set of propositions $\mathcal{P}$.

Syntax

$$f := p \in \mathcal{P} | (f \vee f) | (f \wedge f) | (\neg f) | (f \supset f)$$

Semantics Semantics assigns meaning to a formula. In our case, we wish to assign to each formula a value from $\{\texttt{true}, \texttt{false}\}$. An assignment or a valuation function is a map that assigns to each proposition a truth value i.e. $\mathfrak{v} : \mathcal{P} \mapsto \{\texttt{true}, \texttt{false}\}$. The valuation is extended to formulas as follows. Let $\mathcal{F}$ be the set of all possible propositional formulas and let $\neg\texttt{true} = \texttt{false}$, $\neg\texttt{false} = \texttt{true}$, then $\hat{\mathfrak{v}} : \mathcal{F} \mapsto \{\texttt{true}, \texttt{false}\}$ is defined as

$$
\begin{aligned}
\hat{\mathfrak{v}}(p) &= \mathfrak{v}(p) \text{ for any } p \in \mathcal{P} \\
\hat{\mathfrak{v}}(\neg f) &= \begin{cases} \texttt{true} \text{ if } \hat{\mathfrak{v}}(f) = \\ \texttt{false} \text{ otherwise} \end{cases} \\
\hat{\mathfrak{v}}(f_1 \vee f_2) &= \begin{cases} \texttt{true} \text{ if } \hat{\mathfrak{v}}(f_1) = \texttt{true} \text{ or } \hat{\mathfrak{v}}(f_2) = \texttt{true} \\ \texttt{false} \text{ otherwise} \end{cases} \\
\hat{\mathfrak{v}}(f_1 \wedge f_2) &= \begin{cases} \texttt{true} \text{ if } \hat{\mathfrak{v}}(f_1) = \texttt{true} \text{ and } \hat{\mathfrak{v}}(f_2) = \texttt{true} \\ \texttt{false} \text{ otherwise} \end{cases} \\
\hat{\mathfrak{v}}(f_1 \implies f_2) &= \begin{cases} \texttt{false} \text{ if } \hat{\mathfrak{v}}(f_2) = \texttt{true} \text{ and } \hat{\mathfrak{v}}(f_1) = \texttt{false} \\ \texttt{true} \text{ otherwise} \end{cases}
\end{aligned}
$$

Before we prove any theorems about our logic, we first inductively define the notion of size of a given formula $f \in \mathcal{F}$ as follows.

$$
\texttt{size}(f) = \begin{cases} 0 & \text{if } f = p \text{ for some } p \in \mathcal{P} \\ 1 + \texttt{size}(f_1) & \text{if } f = \neg f_1 \\ \texttt{size}(f_1) + \texttt{size}(f_2) + 1 & \text{if } f = f_1 \ op \ f_2 \end{cases}
$$

With the size defined, we are ready to state and prove the theorem of structural induction. We will extensively use this theorem to prove that a property holds for all formulas. The theorem merely states that in order to prove that a property holds for all formulas, it is sufficient to prove that the property holds for all the propositions and that the property holds for any formula under the assumption that it holds for its sub-formulas.

Theorem 1 (Structural Induction). *To show that a property holds for all formulas, it is sufficient to show*

- *It holds for all atomic propositions $p \in \mathcal{P}$.*
- *Assuming it holds for a formula $f \in \mathcal{F}$, show that it also holds for $\neg f$*
- *Assuming it holds for $f_1, f_2 \in \mathcal{F}$ show that it also holds for $f_1 \ op \ f_2$.*

We define $\texttt{voc}(f)$ for a formula $f \in \mathcal{F}$ as the set of atomic propositions that appear in it.

$$
\begin{aligned}
\texttt{voc}(f) &= \{p\} \text{ if } f = p \in \mathcal{P} \\
\texttt{voc}(f) &= \texttt{voc}(f_1) \text{ if } f = \neg f_1 \\
\texttt{voc}(f) &= \texttt{voc}(f_1) \cup \texttt{voc}(f_2) \text{ if } f = f_1 \ op \ f_2
\end{aligned}
$$

Lemma 1 (Relevance). *Let $f \in \mathcal{F}$ be any formula, let $\mathfrak{v}_1, \mathfrak{v}_2$ be any two valuation such that for all $p \in \texttt{voc}(f)$, $\mathfrak{v}_1(p) = \mathfrak{v}_2(p)$, then $\hat{\mathfrak{v}}_1(f) = \hat{\mathfrak{v}}_2(f)$*

Proof. Base case $f = p \in \mathcal{P}$, then we have nothing to prove.

Suppose $f = \neg f_1$, then by definition, $\texttt{voc}(f) = \texttt{voc}(f_1)$. Then by induction hypothesis, $\hat{\mathfrak{v}}_1(f_1) = \hat{\mathfrak{v}}_2(f_1)$. Now applying the definition of valuation we obtain $\hat{\mathfrak{v}}_1(f) = \hat{\mathfrak{v}}_2(f)$.

Suppose $f = f_1 \vee f_2$, then by definition, $\texttt{voc}(f_1), \texttt{voc}(f_2) \subseteq \texttt{voc}(f)$. Then by induction hypothesis, $\hat{\mathfrak{v}}_1(f_1) = \hat{\mathfrak{v}}_2(f_1)$ and $\hat{\mathfrak{v}}_1(f_2) = \hat{\mathfrak{v}}_2(f_2)$. Hence we have $\hat{\mathfrak{v}}_1(f) = \hat{\mathfrak{v}}_2(f)$. □

Definition 1. *We say a valuation $\mathfrak{v}$ satisfies a formula $f \in \mathcal{F}$ if $\hat{\mathfrak{v}}(f) = \texttt{true}$, we denote it by $\mathfrak{v} \models f$. We say a formula $f \in \mathcal{F}$ is* satisfiable *if there is a valuation $\mathfrak{v}$ such that $\hat{\mathfrak{v}}(f) = \texttt{true}$.*

For a set of formulas $\mathcal{S}$, we say a valuation $\mathfrak{v}$ satisfies $\mathcal{S}$ (denoted $\mathfrak{v} \models \mathcal{S}$) if $\mathfrak{v} \models s$ for each $s \in \mathcal{S}$.

A formula $f \in \mathcal{F}$ is said to be valid *(denoted as $\models f$) if and only if for all valuations $\mathfrak{v}$, $\hat{\mathfrak{v}}(f) = \texttt{true}$. We will sometimes refer to a valid formula as a* tautology*.*

Connections between validity and satisfiability: A formula $f \in \mathcal{F}$ is satisfiable if it is valid, it is valid if and only if $\neg f$ is satisfiable.

Lemma 2. *A formula $f \in \mathcal{F}$ is valid if and only if $\neg f$ is not satisfiable*

Proof. $\neg f$ is not satisfiable $\iff$ for every valuation $\mathfrak{v}, \hat{\mathfrak{v}}(\neg f) = \texttt{false}$ $\iff$ for every $\mathfrak{v}, \hat{\mathfrak{v}}(f) = \texttt{true}$ $\iff$ f is valid. □

2.2 Logical Equivalance

Definition 2. *Given two formulas f_1, f_2, we say they are equivalent, denoted as $f_1 \equiv f_2$ if for every valuation $\mathfrak{v}$, $\hat{\mathfrak{v}}(f_1) = \hat{\mathfrak{v}}(v_2)$. As an example, $p \vee q \equiv q \vee p$.*

Truth table method

One way to prove that two formulas are equivalent is to construct the truth table and show that both the formulas evaluate to the same value on every possible valuation. We already know by means of Relevance Lemma that only those propositions appearing in the formula are relevant.

Application: As an example, truth table method can be used to show that every propositional formula has an equivalent formula in *Disjunctive Normal Form*. A literal is either a proposition or its negation, a DNF clause is simply a conjuction of literals. A DNF formula is one which only contains disjunction of conjunctive clauses. That is, a formula $f \in \mathcal{F}$ is in DNF format if $f = \bigvee_{i=1}^{n} C_i$ for some $n \in \mathbb{N}$, where each C_i is conjunction of literals. Given an arbitrary formula, how to obtain an equivalent DNF formula? Let the given formula be $f \in \mathcal{F}$ and let $\texttt{voc}(f) = \{p_1, \cdots, p_n\}$.

Construct the truth table corresponding the the formula. Notice that each row of the truth table assigns a value to the variable $\{p_1, \cdots, p_n\}$. We will use $\mathfrak{v}_k(p_i)$ to refer to the value that row-k assigns to the propositional variable p_i. We say $\hat{\mathfrak{v}}_k(f) = \texttt{true}$ to mean that the formula f evaluates to true in row-k. Let $S = \{k \mid \hat{\mathfrak{v}}_k(f) = \texttt{true}\}$ be the set of all rows that evaluate f to $\texttt{true}$. We define the clause C_k corresponding to the row k of the truth table as $C_k = \bigwedge_{i=1}^{n} \ell_i$, where $\ell_i = p_i$ if $\mathfrak{v}_k(p_i) = \texttt{true}$ and $\neg p_i$ otherwise. Notice that each C_k uniquely identifies the row-k. Then the DNF formula is got by disjunction over the clauses corresponding to the rows that the formula evalates to $\texttt{true}$. Hence the required DNF formula then is

$$f \equiv \bigvee_{j \in S} C_j$$

Substitution method

Yet another useful tool to show two formulas are equivalent is by *substitution* method that we will describe in sequel.

Definition 3. *Let $p_1, \cdots p_n \in \mathcal{P}$ be a set of propositions, $g_1, \cdots, g_n \in \mathcal{F}$ be a set of formulas. Then a substitution s is defined as a mapping that assigns to each p_i, g_i i.e. $s = [p_1 \leftarrow g_1, \cdots, p_n \leftarrow g_n]$. On applying a substitution s to a formula $f \in \mathcal{F}$, we obtain a new formula as follows*

$$f[s] = \begin{cases} p & \text{if } f = p \in \mathcal{P} \text{ and } p \notin \{p_1, \cdots, p_n\} \\ g_i & \text{if } f = p_i \\ \neg f_1[s] & \text{if } f = \neg f_1 \\ f_1[s] \ op\ f_2[s] & \text{if } f = f_1 \ op\ f_2 \end{cases}$$

The following Substitution-Lemma relates the evaluation of a formula under substitution with the original formula.

Lemma 3. *Let $\mathfrak{v}$ be any valuation, $f \in \mathcal{F}$ any formula and $s = [p_1 \leftarrow g_1, \cdots, p_n \leftarrow g_n]$ a substitution. Then $\hat{\mathfrak{v}}(f[s]) = \hat{\mathfrak{v}}_s(f)$, where $\mathfrak{v}_s$ for any atomic proposition $p \in \mathcal{P}$ is defined as*

$$\mathfrak{v}_s(p) = \begin{cases} \hat{\mathfrak{v}}(g_i) & \text{if } p = p_i \\ \hat{\mathfrak{v}}(p) & \text{otherwise} \end{cases}$$

Proof. Proof by induction.

Suppose $f = p \in \mathcal{P}$ and $p \notin \{p_1, \cdots, p_n\}$ then by definition, $f[s] = p$ and $\mathfrak{v}_s(p) = \hat{\mathfrak{v}}(p)$. Hence $\hat{\mathfrak{v}}(f[s]) = \hat{\mathfrak{v}}_s(f)$

Suppose $f = p_i$ then by definition, $f[s] = g_i$ and $\mathfrak{v}_s(p_i) = \hat{\mathfrak{v}}(g_i)$. Hence $\hat{\mathfrak{v}}(f[s]) = \hat{\mathfrak{v}}_s(f)$

Suppose $f = \neg f_1$ then by definition, $f[s] = \neg f_1[s]$. By induction hypothesis, we have $\hat{\mathfrak{v}}(f_1[s]) = \hat{\mathfrak{v}}_s(f_1)$. Hence we have $\hat{\mathfrak{v}}(f) = \hat{\mathfrak{v}}(\neg f_1[s]) = \hat{\mathfrak{v}}_s(\neg f_1) = \hat{\mathfrak{v}}_s(f)$.

Suppose $f = f_1 \vee f_2$ then by definition, $f[s] = f_1[s] \vee f_2[s]$. By induction hypothesis, we have $\hat{\mathfrak{v}}(f_1[s]) = \hat{\mathfrak{v}}_s(f_1)$ and $\hat{\mathfrak{v}}(f_2[s]) = \hat{\mathfrak{v}}_s(f_2)$. From this, it is easy to see that $\hat{\mathfrak{v}}(f) = \hat{\mathfrak{v}}(f_1[s] \vee f_2[s]) = \hat{\mathfrak{v}}_s(f_1 \vee f_2) = \hat{\mathfrak{v}}_s(f)$.

The other cases are similar.

$\square$

We note that the substitution preserves validity. Let $f \in \mathcal{F}$ be any valid formula and s any substitution, then for any valuation $\mathfrak{v}$, $\mathfrak{v}(f[s]) = \mathfrak{v}_s(f)$ since f is a valid formula, $\mathfrak{v}_s(f) = \texttt{true}$. We will now show that logically equivalent formulas are equivalent under substitution with logically equivalent formulas.

Lemma 4. *Suppose $f \equiv h$ and $g_1 \equiv g_1', g_2 \equiv g_2' \cdots g_n \equiv g_n'$. Let $s = [p_1 \leftarrow g_1, \cdots, p_n \leftarrow g_n]$ and $s' = [p_1 \leftarrow g_1', \cdots, p_n \leftarrow g_n']$ be any two substitutions then, $f[s] \equiv h[s']$.*

Proof. For any valuation $\mathfrak{v}$, we need to prove that $\hat{\mathfrak{v}}(f[s]) = \hat{\mathfrak{v}}(h[s'])$. We prove this in two steps

$f[s] \equiv f[s']$: for any valuation $\mathfrak{v}$, $\hat{\mathfrak{v}}(f[s]) = \hat{\mathfrak{v}}_s(f)$ by substitution lemma. Since for each i, $\mathfrak{v}(g_i) = \mathfrak{v}(g_i')$, we have $\mathfrak{v}_s(p_i) = \mathfrak{v}_{s'}(p_i)$, hence $\mathfrak{v}_s = \mathfrak{v}_{s'}$. From this, we get $\hat{\mathfrak{v}}_s(f) = \hat{\mathfrak{v}}_{s'}(f)$ which implies $f[s] \equiv f[s']$.

$f[s'] \equiv h[s']$: Since $f \equiv h$, we have for any valuation $\mathfrak{v}$, $\hat{\mathfrak{v}}(f) = \hat{\mathfrak{v}}(h)$. Now $\hat{\mathfrak{v}}(f[s']) = \hat{\mathfrak{v}}_{s'}(f) = \hat{\mathfrak{v}}_{s'}(h) = \hat{\mathfrak{v}}(h[s'])$

$\square$

Application: We will now use the above results to show that $\{\vee, \neg\}$ is expressively complete. For this we first define a transformation $\mathcal{T}()$ that transforms an arbitrary formula into one that only uses the above two operations.

$$\begin{aligned}
T(f) &= p && \text{if } f = p \in \mathcal{P}\\
T(f) &= (\neg T(f_1)) && \text{if } f = \neg f_1\\
T(f) &= (T(f_1) \vee T(f_2)) && \text{if } f = f_1 \vee f_2\\
T(f) &= \neg(\neg T(f_1) \vee \neg T(f_2)) && \text{if } f = f_1 \wedge f_2\\
T(f) &= (\neg T(f_1) \vee T(f_2)) && \text{if } f = f_1 \implies f_2
\end{aligned}$$

Lemma 5. *For any formula $f \in \mathcal{F}$,*

$$f \equiv \mathcal{T}(f)$$

Proof. Toward the proof, we need to show that for any valuation $\mathfrak{v}$, $\hat{\mathfrak{v}}(f) = \hat{\mathfrak{v}}(\mathcal{T}(f))$. We do this inductively

Suppose $f = p \in \mathcal{P}$ then $f = \mathcal{T}(f)$ and we have nothing to show.

Suppose $f = \neg f_1$, then inductively we have $f_1 \equiv \mathcal{T}(f_1)$

Suppose $f = f_1 \vee f_2$, then inductively we have $f_1 \equiv \mathcal{T}(f_1))$ and $f_2 \equiv \mathcal{T}(f_2)$ from this the result follows.

Suppose $f = f_1 \wedge f_2$, then inductively we have $f_1 \equiv \mathcal{T}(f_1)$ and $f_2 \equiv \mathcal{T}(f_2)$. Let us assume we have checked that $p \wedge q \equiv \neg(\neg p \vee \neg q)$, then by **??** we have $p \vee \neg q[p \leftarrow f_1, q \leftarrow f_2] \equiv \neg(\neg p \vee \neg q[p \leftarrow \mathcal{T}(f_1), q \leftarrow \mathcal{T}(f_2)]$.

$f = f_1 \implies f_2$, the proof is similar to above case.

□

2.3 Satisfiability

Let us return to the question of when a formula is satisfiable. One easy method is to construct the truth table and check if there is a row that evaluates the formula to `true`. While the running time of this method matches with the best known worst case time, the procedure takes the same time for any formula. Let us convince ourselves that there are fragments of propositional logic for which we can have efficient procedures.

DNF formula: Suppose we are guaranteed that the formula given to us is in DNF format, we only need to find in it a clause that is satisfiable. A conjunctive clause is satisfiable if and only if both p and $\neg p$ do not appear in it. Hence we have a linear time procedure for checking satisfiability of a DNF formula.

Horns Formula: We say a disjunctive clause is a Horn clause if it contains at-most one positive literal. For example, $(p_1 \vee \neg p_2 \vee \neg p_3)$ is a Horn clause. A Horn formula is a CNF formula with only Horn clauses. As an example, $(p_1 \vee \neg p_2 \vee \neg p_3) \wedge (p_2 \vee \neg p_1 \vee \neg p_3) \wedge (p_3) \wedge (\neg p_4)$ is a Horn formula. A Horn formula can be written using implications in an intuitive way as follows

$$((p_2 \wedge p_3) \implies p_1) \wedge ((p_1 \wedge p_3) \implies p_2) \wedge (\texttt{true} \implies p_3) \wedge (p_4 \implies \texttt{false})$$

We now describe a polynomial time procedure that decides whether a given Horn formula is satisfiable or not. In fact it produces a valuation in case it is satisfiable.

Termination & Correctness: Termination is easy to see. In each iteration, the procedure picks an unsatisfied clause $\alpha \implies \beta$. If $\beta = \texttt{false}$ then it returns UNSAT. Otherwise, β is a propositional variable and $A(\beta) = \texttt{false}$. The algorithm flips this value to true. Since there are only finitely many flips that can happen, the algorithm has to terminate. For correctness, we note that if the procedure returns SAT then it also produces a satisfying valuation hence f is indeed satisfiable. We have to prove that in case f is satisfiable, then the algorithm outputs SAT. For this, we observe that if $\mathfrak{v} \models f$, then for all $p \in \{p_1, \cdots, p_n\}$, we have $A[p] \leq \mathfrak{v}(p)$ after each iteration. Here we consider $\texttt{false} < \texttt{true}$. Clearly the invariant is true at the start, suppose there is a unsatisfied clause of the form $r_1 \wedge \cdots \wedge r_\ell \implies r$ then

Algorithm 1 Satisfiability of Horn formula

```
 1: procedure HORN($f \in \mathcal{F}$)                          ▷ $f$ is given to be a Horn formula
 2:     Let $\texttt{voc}(f) = \{p_1, \cdots, p_n\}$
 3:     Let $A[p_1, \cdots, p_n] = 0$ be an array with one cell per variable in $\texttt{voc}(f)$
 4:     while $A$ does not satisfy $f$ do
 5:         Pick an clause $\alpha \implies \beta$ that is not satisfied
 6:         if $\beta = \texttt{false}$ then                              ▷ $\beta$ is not a variable
 7:             return UNSAT
 8:         end if
 9:         $A[\beta] = \texttt{true}$                                     ▷ $\beta$ is a variable
10:     end while
11:     return SAT
12: end procedure
```

inductively $A[r_i] = \mathfrak{v}(r_i) = \texttt{true}$. Further we have $A[r] = \texttt{false}$ and $\mathfrak{v}(r) = \texttt{true}$. Hence the invariant holds after the procedure flips $A[r]$. From this it is clear that if f is satisfiable then there is a way to obtain a $A \leq \mathfrak{v}$.

Semantic Tableu

Semantic Tableau is an efficient procedure to decide if a formula is satisfiable. The idea here is to search for a model by decomposing the formula into set of literals. For this, we define two sets of formulas namely α-formulas and β-formulas. The procedure constructs a tree iteratively. Initially there is the root node that is labelled with the given formula. Each node that is created is either marked or unmarked. In each iteration, an unmarked node is picked. Then a formula the the node is picked. If the formula is an α-formula then a duplicate node is added as a child and the α-formula is replaced with the corresponding $\alpha_1 \cup \alpha_2$. If the formula is a β-formula, then node is duplicated twice and added as left and right child. In the left child, the formula is replaced with β_1 and in the right child, it is replaced with β_2. We describe this procedure below.

α-Formula	α_1	α_2
$\neg\neg A$	A	A
$A_1 \wedge A_2$	A_1	A_2
$\neg(A_1 \vee A_2)$	$\neg A_1$	$\neg A_2$
$\neg(A_1 \implies A_2)$	A_1	$\neg A_2$

β-Formula	β_1	β_2
$\neg(A_1 \wedge A_2)$	$\neg A_1$	$\neg A_2$
$A_1 \vee A_2$	A_1	A_2
$(A_1 \implies A_2)$	$\neg A_1$	A_2

Termination: Intuitively it is clear that the procedure terminates since in each iteration, a formula is replaced by a shorter variant. Towards a formal proof, we associate a $\texttt{wt}()$ weight function with each node and show that the weight function strictly decreases as we go down the tree. For any formula $f \in \mathcal{F}$, we define the weight function as

Algorithm 2 Semantic Tableau procedure

```
1: procedure SemanticTableau($f \in \mathcal{F}$)
2:     Create a root node $r$ and label it with $\{f\}$ i.e. $\texttt{Lab}(r) = \{f\}$.
3:     Suppose $f = p \in \mathcal{P}$ then mark it with a ✓.    ▷ Since the node has only literals, we mark it
4:     while there is an unmarked leaf $t$ in the tree do
5:         Then $t$ contains either an $\alpha$ or a $\beta$ formula say $\phi$
6:         if $\phi$ is an $\alpha$ formula then
7:             Create a child node $t_1$ and let $\texttt{Lab}(t_1) = \texttt{Lab}(t) \setminus \{\phi\} \cup \alpha_1(\phi) \cup \alpha_2(\phi)$
8:         end if
9:         if $\phi$ is a $\beta$ formula then
10:            Create two childrens $t_1, t_2$
11:            Let $\texttt{Lab}(t_1) = \texttt{Lab}(t) \setminus \{\phi\} \cup \beta_1(\phi)$
12:            Let $\texttt{Lab}(t_2) = \texttt{Lab}(t) \setminus \{\phi\} \cup \beta_2(\phi)$
13:        end if
14:        if $t_1$ only contains literals then
15:            if $\texttt{Lab}(t_1)$ does not contain $\{p, \neg p\}$ for any $p \in \mathcal{P}$ then
16:                Mark $t_1$ as ✓
17:            else
18:                Mark $t_1$ as ✗
19:            end if
20:        end if
21:        if $t_2$ only contains literals then
22:            if $\texttt{Lab}(t_2)$ does not contain $\{p, \neg p\}$ for any $p \in \mathcal{P}$ then
23:                Mark $t_2$ as ✓
24:            else
25:                Mark $t_2$ as ✗
26:            end if
27:        end if
28:    end while
29:    return SAT
30: end procedure
```

$$\begin{aligned}
\texttt{wt}(f) &= 1 && \text{if } f = p \in \mathcal{P} \\
\texttt{wt}(\neg f_1) &= 1 + \texttt{wt}(f_1) \\
\texttt{wt}(f_1 \ op \ f_2) &= 2 + \texttt{wt}(f_1) + \texttt{wt}(f_2)
\end{aligned}$$

For example $\texttt{wt}(\neg p \vee \neg q) = 7$. We define the weight of a node t as $\texttt{wt}(t) = \Sigma_{\phi \in \texttt{Lab}(t)} \texttt{wt}(t)$. Note that the notion of weight is very similar to the notion of size that we have already seen, we re-define it in this way for a perticular reason. We will get to this in a minute. We will show that in every iteration, the weight of the child that we newly introduce decreases. Suppose the formula ϕ chosen was an α-formula and $\phi = (A_1 \wedge A_2)$, then

$$\texttt{wt}(t_1) = \texttt{wt}(t) - (\texttt{wt}(A_1) + 2 + \texttt{wt}(A_2)) + \texttt{wt}(A_1) + \texttt{wt}(A_2) = \texttt{wt}(t) - 2$$

Suppose the formula ϕ chosen was an α-formula and $\phi = \neg(A_1 \vee A_2)$, then

$$\texttt{wt}(t_1) = \texttt{wt}(t) - (1 + \texttt{wt}(A_1) + 2 + \texttt{wt}(A_2)) + 1 + \texttt{wt}(A_1) + 1 + \texttt{wt}(A_2) = \texttt{wt}(t) - 1$$

Notice that in the above case, we need that the weight of binary operators to be more than the weight of the negation operator. Otherwise, we will not have a strictly reducing weight. The result for rest of the formulas can easily be verified.

Soundness and Completeness: Let $\mathcal{T}$ be the complete tree that the procedure produces when it terminates. We say it is closed if every leaf in it is marked as ✗, it is open otherwise. The correctness of the procedure follows from the following theorem.

Theorem 2. *Let $\mathcal{T}$ be a complete tableau for a formula f, then f is unsatisfiable if and only if $\mathcal{T}$ is closed.*

Soundness We will first prove that if $\mathcal{T}$ is closed then f is UNSAT. To prove this, we show that for any node t in the tableau, it the tree sub-tending from it is closed then $\lambda(t)$ is UNSAT. We prove this by inducting on the height of the tableau.

- Case height $= 0$, then clearly $\{p, \neg p\} \subseteq$ and the result is immediate.
- Case height > 0, Suppose t has a unique child t', then some α formula was used to create the child. Let $\texttt{Lab}(t) = U \cup \{h\}$ where h was the α formula that was chosen. Then $\texttt{Lab}(t') = U \cup \{\alpha_1(h), \alpha_2(h)\}$. Inductively, $\texttt{Lab}(t')$ is UNSAT. Which means either U is UNSAT or $\{\alpha_1(h), \alpha_2(h)\}$ is UNSAT. Using this information one can easily argue that $\texttt{Lab}(t)$ is also UNSAT.

  Suppose t has two children, the argument is very similar.

Completeness In order to show completeness, we first define what are called the *Hintikka* sets. A set of formulas S is said to be Hintikka if it satisfies the following properties.

- $\{p, \neg p\} \not\subseteq S$
- if an α formula $h \in S$ then $\alpha_1(h) \in S$ and $\alpha_2(h) \in S$
- if an β formula $h \in S$ then $\beta_1(h) \in S$ and $\beta_2(h) \in S$

We prove that any Hintikka set is satisfiable.

Lemma 6. *Let s be a Hintikka set, then S is satisfiable.*

Proof. We define a valuation function $\mathfrak{v}$ as

$$\forall p \in \mathcal{P}, \mathfrak{v}(p) = \texttt{false} \iff \neg p \in S$$

We need to show that $\mathfrak{v} \models S$. Let $h \in S$ be any formula, if h is a literal then it is straight forward to see that $\mathfrak{v}(h) = \texttt{true}$. If h is an α formula and suppose $h = h_1 \wedge h_2$. Then $h_1, h_2 \in S$ and inductively we have $\mathfrak{v}(h_1) = \mathfrak{v}(h_2) = \texttt{true}$, from this we can conclude $\mathfrak{v}(h) = \texttt{true}$. The other cases are similar. $\square$

To prove completeness, we prove that if tableau $\mathcal{T}$ is open then f is satisfiable. Note that suppose $\mathcal{T}$ is open, then there is at-least one leaf node that is open. Collect all the formulas that appear in the path from this leaf node to the root, we can show that this set is Hintikka. to do rewrite the proof with details

Excercise 1. *Show that the set of all for propositional formulas is countable.*

2.4 Compactness

Suppose we are given an infinite set of propositional formulas, what can we say about its satisfiability? Compactness theorem states that if the infinite set is `UNSAT` then it has a finite witness. The other way of looking at it is any set of propositional formula is satisfiable if and only if there is no finite witness for its unsatisfiability. We call a set of formulas *finitely satisfiable* if every finite subset of it is satisfiable. That is, a set of formulas Γ is finitely satisfiable if every $\Gamma_0 \subseteq_{\texttt{fin}} \Gamma$, Γ_0 is satisfiable. Compactness theorem says that a set is satisfiable if and only if it is finitely satisfiable.

Theorem 3. *A set Γ is satisfiable if and only if it is finitely satisfiable.*

Notice that one direction is trivial, if a set is satisfiable then every subset (finite or not) of it is also satisfiable. We now show how to prove the other direction. The proof strategy would be as follows, we extend Γ that is finitely satisfiable into a larger set Γ_H which is also finitely satisfiable. We show that the larger set is a Hintikka set. Recall we proved already that any Hintikka set is satisfiable and hence also Γ_H. This automatically will also prove that $\Gamma \subseteq \Gamma_H$ is satisfiable.

We construct a sequence of set of formulas $\Gamma = \Gamma_0 \subseteq \Gamma_1 \subseteq \Gamma_2 \cdots$ and define $\Gamma_H = \bigcup_{i \in \mathbb{N}} \Gamma_i$. During the construction, we ensure that each Γ_i is finitely satisfiable. We already proved in our exercise 1 that the set of all propositional formulas is countable. From this, we can assume that there is an ordering $\leq$ among all the propositional formulas. We construct the sequence of set of formulas in rounds. In round zero, we let $\Gamma_0 = \Gamma$. In a round $i+1$, we will assume that we already have Γ_i and construct Γ_{i+1}. We will pick an unmarked h that is the least $\leq$ formula in Γ_i mark it and construct Γ_{i+1} as

$$\Gamma_{i+1} = \begin{cases} \Gamma_i & \text{if } h \text{ is a literal} \\ \Gamma_i \cup \{\alpha_1(h), \alpha_2(h)\} & \text{if } h \text{ is a } \alpha \text{ formula} \\ \Gamma_i \cup \beta_1(h) & \text{if } h \text{ is a } \beta \text{ formula and } \Gamma_i \cup \beta_1(h) \text{ is finitely satisfiable} \\ \Gamma_i \cup \beta_2(h) & \text{otherwise} \end{cases}$$

Claim 1. *For each $i \in \mathbb{N}$, we have Γ_i is finitely satisfiable.*

Proof. For $i = 0$, we have nothing to prove. Let $i > 0$, we will assume Γ_{i-1} is finitely satisfiable and prove that so is Γ_{i+1}. Let h be the least $\leq$ formula.

Suppose h was a literal then by construction, $\Gamma_{i+1} = \Gamma_i$ and by IH, we have $\Gamma_i = \Gamma_{i+1}$ is finitely satisfiable.

Suppose h was an α formula, then $\Gamma_{i+1} = \Gamma_i \cup \{\alpha_1(h), \alpha_2(h)\}$. We need to show that any arbitrary subset $\Gamma' \subseteq_{\texttt{fin}} \Gamma_{i+1}$ is satisfiable. Suppose $\Gamma' \subseteq_{\texttt{fin}} \Gamma_i$, then clearly it is satisfiable, let us assume $\{\alpha_1(h), \alpha_2(h)\} \cap \Gamma' \neq \emptyset$. Let $\mathcal{S} = \Gamma' \setminus \{\alpha_1(h), \alpha_2(h)\}$. Notice that $S \subseteq_{\texttt{fin}} \Gamma_i$, so is $\mathcal{S} \cup \{h\} \subseteq_{\texttt{fin}} \Gamma_i$. From this, we can deduce that $\mathcal{S} \cup \{h\}$ is satisfiable. We also know that $h \equiv \alpha_1(h) \wedge \alpha_2(h)$, hence $\mathcal{S} \cup \{h, \alpha_1(h), \alpha_2(h)\}$ is also satisfiable. This also implies that $\Gamma' \subseteq \mathcal{S} \cup \{h, \alpha_1(h), \alpha_2(h)\}$ is also satisfiable.

Suppose h was a β formula, then $\Gamma_{i+1} = \Gamma_i \cup \beta$ where $\beta \in \{\beta_1(h), \beta_2(h)\}$. Now we want to show that $\Gamma_i \cup \beta$ is satisfiable. Let us suppose that it is not finitely satisfiable, then clearly neither $\Gamma_i \cup \{\beta_1(h)\}$ nor $\Gamma_i \cup \{\beta_2(h)\}$ is finitely satisfiable, this means that there is a $\Gamma'_1 \subseteq_{\texttt{fin}} \Gamma_i \cup \{\beta_1(h)\}$ and there is a $\Gamma'_2 \subseteq_{\texttt{fin}} \Gamma_i \cup \{\beta_2(h)\}$ such that both Γ'_1, Γ'_2 are UNSAT. But notice that $\Gamma' = \Gamma'_1 \cup \Gamma'_2 \setminus \{\beta_1(h), \beta_2(h)\} \subseteq_{\texttt{fin}} \Gamma_i$ and hence is satisfiable. Infact $\Gamma' \cup \beta(h) \subseteq_{\texttt{fin}} \Gamma_i$ is also satisfiable. But we know that $\beta(h) \equiv \beta_1(h) \vee \beta_2(h)$. This means either $\Gamma' \cup \{\beta_1(h)\}$ or $\Gamma' \cup \{\beta_2(h)\}$ is satisfiable. But then it contradicts the fact that both $\Gamma'_1 \subseteq \Gamma' \cup \{\beta_1(h)\}$ and $\Gamma'_2 \subseteq \Gamma' \cup \{\beta_2(h)\}$ are UNSAT. □

Corollary 1. *Γ_H is finitely satisfied.*

Proof. If it is not, then there is a $\gamma' \subseteq_{\texttt{fin}} \Gamma_H$ that is not satisfied. But this $\Gamma' \in \Gamma_\ell$ for some $\ell \in \mathbb{N}$. But we have shown that for all $i \in \mathbb{N}$, Γ_i is finitely satisfied. □

We now let $\Gamma_H = \bigcup_{i \in \mathbb{N}} \Gamma_i$ and prove that this set is Hintikka.

Lemma 7. *Γ_H is a Hintikka set.*

Proof. For this, we need to show that 1) for any α formula $h \in \Gamma_H$, $\{\alpha_1(h), \alpha_2(h)\} \subseteq \Gamma_H$, 2) for any β formula $h \in \Gamma_H$, $\{\beta_1(h), \beta_2(h)\} \cap \Gamma_H \neq \emptyset$ and 3) that for any $p \in \mathcal{P}$, $\{p, \neg p\} \not\subseteq \Gamma_H$.

For $1, 2$, we simply note that there are only finitely many formulas f such that $f \leq h$. Hence if $h \in \Gamma_H$ then there is a ℓ such that f will be the least unmarked formula in Γ_ℓ. By construction, the requirement will be satisfied.

To show 3, we simply note that if $\{p, \neg p\} \subseteq \Gamma_H$ then Γ_H is not finitely satisfied. But we already have shown that Γ_H is finitely satisfied. □

2.4.1 Applications of Compactness

k-colorability: We say a graph $\mathcal{G} = (V, E)$ is k colourable if we can colour the vertices of the graph with at-most k many colors such that no two adjacent vertices have the same colours. As the first application of compactness, we show that a graph $\mathcal{G}$ is k-colourable if and only if every finite sub-graph of it is k-colourable.

To this end, we will first show how to represent k-colorability of a given graph as a set of propositional formulas. Notice that the set of formulas has to ensure the following two points.

1. That each vertex is assigned exactly one colour
2. That the adjacent vertices are not assigned the same colour.

Let $\mathcal{G} = (V, E)$ be the given graph and $[1, k]$ be the set of colours. We first fix the set of propositional variables that our set of formulas will be using. We will use $\{p_{uj} \mid u \in V, j \in [1, k]\}$ as the set of propositions. Intuitively p_{uj} asserts that the vertex $u \in V$ is coloured with the colour j. For any $u \in V$, we define $\texttt{Col}_k(u)$ as

$$\texttt{Col}_k(u) = \bigvee_{j=1}^{k} p_{uj} \quad \wedge \quad \bigwedge_{j \neq \ell} \neg(p_{uj} \wedge p_{u\ell})$$

Now for any graph $\mathcal{G}$, we define $\texttt{Col}_k(\mathcal{G}) = \bigcup_{u \in V} \texttt{Col}(u)$. Notice that $\texttt{Col}(\mathcal{G})$ ensures that each vertex in the graph is coloured by exactly one colour. We next show how to build the set of formulas that ensures that adjacent vertices are not labelled with the same color. Given an edge $(u, v) \in E$, we define $\texttt{edgeCon}_k(u, v)$ as

$$\texttt{edgeCon}_k(u, v) = \bigwedge_{\ell \in [1,k]} \neg(p_{u\ell} \wedge p_{v\ell})$$

For any graph $\mathcal{G}$, we define $\texttt{edgeCon}_k(\mathcal{G}) = \bigcup_{(u,v) \in E} \texttt{edgeCon}(u, v)$. We do not impose any restrictions on the graph itself, it may be an infinite graph. The following claim characterises k colorability of a graph using a set of propositional formulas.

Claim 2. *A graph $\mathcal{G} = (V, edges)$ is k-colourable if and only if $\texttt{Col}_k(\mathcal{G}) \cup \texttt{edgeCon}_k(\mathcal{G})$ is satisfiable*

We will now use the compactness theorem to prove the following theorem which states that a graph is k colourable if and only if every finite sub-graph of it is.

Theorem 4. *A graph $\mathcal{G}$ is k-colourable if and only if every finite induced sub-graph of it is k-colourable.*

Proof. One direction is trivial, if a graph is k-colourable then indeed every finite induced sub-graph of it is also k-colourable. We will prove the other direction. That is assuming that every finite sub-graph of a graph is colourable, we will show that the whole graph is also k-colourable. But notice that for this, it is enough to show that $\Gamma = \mathtt{Col}_k(\mathcal{G}) \cup \mathtt{edgeCon}_k(\mathcal{G})$ is satisfiable. In fact it is enough to show that every finite subset of Γ is satisfiable.

Let $S \subseteq_{\mathtt{fin}} \Gamma$ be a finite subset of Γ, let V_S be the set of all vertices that appear in the formulas of S. For example, u appears in $\mathtt{Col}_k(u)$ and u, v appears in $\mathtt{edgeCon}_k(u, v)$. Let $\mathcal{G}_{V_S}$ be the sub-graph induced by V_S, let $T = \mathtt{Col}_k(\mathcal{G}_{V_S}) \cup \mathtt{edgeCon}_k(\mathcal{G}_{V_S})$, clearly $S \subseteq T$. Notice that $\mathcal{G}_{V_S}$ is k-colourable since it is a finite sub-graph of $\mathcal{G}$. This also implies that T and hence S is satisfiable. □

In-expressiability: We will now show that there are set of valuation functions that cannot be represented by any set of propositional formulas. Let $\mathcal{P} = \{p_1, p_2, p_3, \cdots\}$ be the set of all propositional variables, we will assume that these propositions are ordered by the index.We let $\mathfrak{v}_i$ to be the valuation function that maps $\{p_1, \cdots, p_i\}$ to $\mathtt{true}$ and the rest to $\mathtt{false}$. Consider the following set of valuation functions

$$\mathcal{S} = \{v_i \mid i \in \mathbb{N}\}$$

We claim that there is no set of formulas Γ which are only satisfiable by the valuations in $\mathcal{S}$.

Theorem 5. *There is no set of formulas $\Gamma \subseteq \mathcal{F}$ such that $\mathfrak{v} \models \Gamma$ if and only if $\mathfrak{v} \in \mathcal{S}$*

Proof. Let us assume to the contrary that there is such a Γ, let $\Gamma' = \Gamma \cup \mathcal{P}$. We first note that the only valuation that satisfies $\mathcal{P}$ is $\mathfrak{v}_{\mathtt{true}}$ which maps every proposition to $\mathtt{true}$. However $\mathfrak{v}_{\mathtt{true}} \notin \mathcal{S}$ and hence $\mathfrak{v}_{\mathtt{true}} \not\models \Gamma$. So we have that Γ' is $\mathtt{UNSAT}$.

We will now also establish that Γ' is satisfiable, by using our compactness theorem. Let $T \subseteq_{\mathtt{fin}} \Gamma'$ be any finite subset, then T can be decomposed as $T = T_\Gamma \cup T_\mathcal{P}$ where $T_\Gamma \subseteq \Gamma$ and $T_\mathcal{P} \subseteq \mathcal{P}$. Let $p_{\mathtt{max}}$ be the variable with maximum index appearing in $T_\mathcal{P}$. Clearly $\mathfrak{v}_{\mathtt{max}} \models T_\mathcal{P}$, $\mathfrak{v}_{\mathtt{max}} \in \mathcal{S}$ and hence $\mathfrak{v}_{\mathtt{max}} \models T$. Since we chose an arbitrary finite subset T, we have that every finite subset of Γ' is satisfiable. By compactness theorem, we also have that Γ' is satisfiable, a contradiction. □