**Outline**

- Recognizing Well-Formed Formulas
- Propositional Logic: Semantics
- Truth Tables
- Satisfiability and Tautologies
- Propositional Connectives and Boolean Functions
- Compactness
- Computability and Decidability

Sources:
Enderton, Sections: 1.2, 1.3, 1.5, 1.7.

N. J. Cutland. *Computability*.


**Propositional Logic: Well-Formed Formulas**

Recall our inductive definition of the set \( W \) of well-formed formulas in propositional logic. Given the alphabet \( \{(), \neg, \land, \lor, \rightarrow, \leftrightarrow, A_1, A_2, \ldots \} \),

- \( U \) = the set of all expressions over the alphabet.
- \( B \) = the set of expressions consisting of a single propositional symbol.
- \( F \) = the set of formula-building operations:
  - \( \mathcal{E}_\neg(\alpha) = (\neg\alpha) \)
  - \( \mathcal{E}_\land(\alpha, \beta) = (\alpha \land \beta) \)
  - \( \mathcal{E}_\lor(\alpha, \beta) = (\alpha \lor \beta) \)
  - \( \mathcal{E}_\rightarrow(\alpha, \beta) = (\alpha \rightarrow \beta) \)
  - \( \mathcal{E}_\leftrightarrow(\alpha, \beta) = (\alpha \leftrightarrow \beta) \)
An Algorithm for Recognizing WFFs

**Lemma**

Let $\alpha$ be a wff. Then exactly one of the following is true.

- $\alpha$ is a propositional symbol.
- $\alpha = (\neg \beta)$ where $\beta$ is a wff.
- $\alpha = (\beta \odot \gamma)$ where $\odot$ is one of $\{\land, \lor, \rightarrow, \leftrightarrow\}$, $\beta$ is the first parentheses-balanced initial segment of the result of dropping the first ( from $\alpha$, and $\beta$ and $\gamma$ are wffs.

**How would you prove this?**

Induction, of course!

---

**An Algorithm for Recognizing WFFs**

**Termination**

*How do we prove termination of this algorithm?*

We can show that the sum of the lengths of all the expressions labeling leaves decreases on each iteration of the loop.

**Soundness**

If the algorithm returns true when given input $\alpha$, then $\alpha$ is a wff.

The proof is by induction on the tree $T$ generated by the algorithm from the leaves up to the root.

**Completeness**

If $\alpha$ is a wff, then the algorithm will return true.

Proof using the induction principle for the set of wffs.

---

**Notational Conventions**

- Larger variety of propositional symbols: $A, B, C, D, p, q, r$, etc.
- Outermost parentheses can be omitted: $A \land B$ instead of $(A \land B)$.
- Negation symbol binds stronger than binary connectives and its scope is as small as possible: $\neg A \land B$ means $((\neg A) \land B)$.
- $\{\land, \lor\}$ bind stronger than $\{\rightarrow, \leftrightarrow\}$: $A \land B \rightarrow \neg C \lor D$ is $((A \land B) \rightarrow ((\neg C) \lor D))$.
- When one symbol is used repeatedly, grouping is to the right: $A \land B \land C$ is $(A \land (B \land C))$.

Note that conventions are only unambiguous for wffs, not for arbitrary expressions.
Propositional Logic: Semantics

Intuitively, given a wff $\alpha$ and a value (either $T$ or $F$) for each propositional symbol in $\alpha$, we should be able to determine the value of $\alpha$.

How do we make this precise? Let $v$ be a function from $B$ to $\{F, T\}$. We call this function a truth assignment.

Now, we define $v_\alpha$, a function from $W$ to $\{F, T\}$ as follows (we compute with $F$ and $T$ as if they were 0 and 1 respectively).

• For each propositional symbol $A_i$, $v(A_i) = v(A_i)$.
• $v(\neg(\alpha)) = T - v(\alpha)$
• $v(\land(\alpha, \beta)) = \min(v(\alpha), v(\beta))$
• $v(\lor(\alpha, \beta)) = \max(v(\alpha), v(\beta))$
• $v(\rightarrow(\alpha, \beta)) = \max(T - v(\alpha), v(\beta))$
• $v(\leftrightarrow(\alpha, \beta)) = T - |v(\alpha) - v(\beta)|$

The recursion theorem and the unique readability theorem guarantee that $\tau$ is well-defined.

Complex truth tables

Truth tables can also be used to calculate all possible values of $v$ for a given wff. We associate a column with each propositional symbol and a column with each propositional connective. There is a row for each possible truth assignment to the propositional connectives.

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$(A_1 \lor A_2) \land \neg A_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T  T  T  T  F  F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T  T  T  T  T  T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T  F  F  F  F  T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T  F  F  F  F  F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F  F  F  F  F  T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F  F  F  F  F  T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F  F  F  F  F  F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F  F  F  F  T  T</td>
</tr>
</tbody>
</table>

Definitions

If $\alpha$ is a wff, then a truth assignment $v$ satisfies $\alpha$ if $v(\alpha) = T$.

A wff $\alpha$ is satisfiable if there exists some truth assignment $v$ which satisfies $\alpha$.

Suppose $\Sigma$ is a set of wffs. Then $\Sigma$ tautologically implies $\alpha$, $\Sigma \models \alpha$, if every truth assignment which satisfies each formula in $\Sigma$ also satisfies $\alpha$.

Particular cases:

• If $\emptyset \models \alpha$, then we say $\alpha$ is a tautology or $\alpha$ is valid and write $\models \alpha$.
• If $\Sigma$ is unsatisfiable, then $\Sigma \models \alpha$ for every wff $\alpha$.
• If $\alpha \models \beta$ (shorthand for $\{\alpha\} \models \beta$) and $\beta \models \alpha$, then $\alpha$ and $\beta$ are tautologically equivalent.
• $\Sigma \models \alpha$ if and only if $\land(\Sigma) \rightarrow \alpha$ is valid.
Examples

- \((A \lor B) \land (\neg A \lor \neg B)\) is satisfiable, but not valid.
- \((A \lor B) \land (\neg A \lor \neg B) \land (A \leftrightarrow B)\) is unsatisfiable.
- \(\left\{A, A \rightarrow B\right\} \models B\)  
  \(A \land (A \rightarrow B) \land (\neg B)\)
- \(\left\{A, \neg A\right\} \models (A \land (\neg A) \land (\neg A \land \neg A))\)
- \((A \land B)\) is tautologically equivalent to \(\neg A \lor \neg B\)
  \(\neg ((A \land B) \leftrightarrow (\neg A \lor \neg B))\)

Suppose you had an algorithm \(\text{SAT}\) which would take a \(\text{wff} \ \alpha\) as input and return \text{true} if \(\alpha\) is satisfiable and \text{false} otherwise. How would you use this algorithm to verify each of the claims made above?

Determining Satisfiability using Truth Tables

An Algorithm for Satisfiability

To check whether \(\alpha\) is satisfiable, form the truth table for \(\alpha\). If there is a row in which \(T\) appears as the value for \(\alpha\), then \(\alpha\) is satisfiable. Otherwise, \(\alpha\) is unsatisfiable.

An Algorithm for Tautological Implication

To check whether \(\left\{\alpha_1, \ldots, \alpha_k\right\} \models \beta\), check the satisfiability of \((\alpha_1 \land \ldots \land \alpha_k) \land (\neg \beta)\). If it is unsatisfiable, then \(\left\{\alpha_1, \ldots, \alpha_k\right\} \models \beta\), otherwise \(\left\{\alpha_1, \ldots, \alpha_k\right\} \not\models \beta\).

Determining Satisfiability using Truth Tables

Example

\(A \land ((B \lor \neg A) \land (C \lor \neg B))\)

<table>
<thead>
<tr>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(A \land ((B \lor \neg A) \land (C \lor \neg B)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F)</td>
<td>(F)</td>
<td>(F)</td>
<td>(F)</td>
</tr>
<tr>
<td>(F)</td>
<td>(F)</td>
<td>(T)</td>
<td>(F)</td>
</tr>
<tr>
<td>(F)</td>
<td>(T)</td>
<td>(F)</td>
<td>(F)</td>
</tr>
<tr>
<td>(F)</td>
<td>(T)</td>
<td>(T)</td>
<td>(F)</td>
</tr>
<tr>
<td>(T)</td>
<td>(F)</td>
<td>(F)</td>
<td>(F)</td>
</tr>
<tr>
<td>(T)</td>
<td>(T)</td>
<td>(F)</td>
<td>(F)</td>
</tr>
<tr>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
</tr>
</tbody>
</table>

Example

\((A \lor B) \land (\neg A \lor \neg B)\) is satisfiable, but not valid.
\((A \lor B) \land (\neg A \lor \neg B) \land (A \leftrightarrow B)\) is unsatisfiable.
\(\left\{A, A \rightarrow B\right\} \models B\)  
\((A \land (A \rightarrow B) \land (\neg B))\)
\(\left\{A, \neg A\right\} \models (A \land (\neg A) \land (\neg A \land \neg A))\)
\((A \land B)\) is tautologically equivalent to \(\neg A \lor \neg B\)
\(\neg ((A \land B) \leftrightarrow (\neg A \lor \neg B))\)
Determining Satisfiability using Truth Tables

What is the complexity of this algorithm?

$2^n$ where $n$ is the number of propositional symbols.

Can you think of a way to speed up these algorithms?

In an upcoming lecture, we will discuss some of the applications and best-known techniques for the SAT algorithm.

Some tautologies

### Associative and Commutative laws for $\land, \lor, \leftrightarrow$

### Distributive Laws

- $(A \land (B \lor C)) \leftrightarrow ((A \land B) \lor (A \land C))$
- $(A \lor (B \land C)) \leftrightarrow ((A \lor B) \land (A \lor C))$

### Negation

- $\neg\neg A \leftrightarrow A$
- $(A \rightarrow B) \leftrightarrow (A \land \neg B)$
- $(A \leftrightarrow B) \leftrightarrow ((A \land \neg B) \lor (\neg A \land B))$

### De Morgan’s Laws

- $\neg (A \land B) \leftrightarrow (\neg A \lor \neg B)$
- $\neg (A \lor B) \leftrightarrow (\neg A \land \neg B)$

More Tautologies

**Implication**

- $(A \rightarrow B) \leftrightarrow (\neg A \lor B)$

**Excluded Middle**

- $A \lor \neg A$

**Contradiction**

- $\neg (A \land \neg A)$

**Contraposition**

- $(A \rightarrow B) \leftrightarrow (\neg B \rightarrow \neg A)$

**Exportation**

- $((A \land B) \rightarrow C) \leftrightarrow (A \rightarrow (B \rightarrow C))$

Propositional Connectives

We have five connectives: $\neg, \land, \lor, \rightarrow, \leftrightarrow$. Would we gain anything by having more? Would we lose anything by having fewer?

**Example: Ternary Majority Connective $\#$**

$E_{\#}(\alpha, \beta, \gamma) = (\#\alpha\beta\gamma)$

$\overline{\nu}(\#\alpha\beta\gamma) = T$ iff the majority of $\overline{\nu}(\alpha), \overline{\nu}(\beta),$ and $\overline{\nu}(\gamma)$ are $T$.

What does this new connective do for us?

**Claim:** The extended language obtained by allowing this new symbol has the same expressive power as the original language.

How do we show this formally?
Boolean Functions

For \( k \geq 0 \), a \( k \)-place Boolean function is a function from \( \{\text{F}, \text{T}\}^k \) to \( \{\text{F}, \text{T}\} \). A Boolean function then is anything which is a \( k \)-place Boolean function for some \( k \).

Each \( \alpha \) determines a corresponding Boolean function \( B_\alpha \). For example, if \( \alpha \equiv A_1 \land A_2 \), then \( B_\alpha \) is a 2-place Boolean function whose value is given by the following table.

<table>
<thead>
<tr>
<th>( X_1 )</th>
<th>( X_2 )</th>
<th>( B_\alpha(X_1, X_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{T} )</td>
<td>( \text{T} )</td>
<td>( \text{T} )</td>
</tr>
<tr>
<td>( \text{T} )</td>
<td>( \text{F} )</td>
<td>( \text{F} )</td>
</tr>
<tr>
<td>( \text{F} )</td>
<td>( \text{T} )</td>
<td>( \text{F} )</td>
</tr>
<tr>
<td>( \text{F} )</td>
<td>( \text{F} )</td>
<td>( \text{F} )</td>
</tr>
</tbody>
</table>

Realizing Boolean Functions

In general, suppose that \( \alpha \) is a wff whose propositional symbols are included in \( A_1, \ldots, A_n \). We define an \( n \)-place Boolean function \( B_\alpha^n \), the Boolean function realized by \( \alpha \) as

\[
B_\alpha^n(X_1, \ldots, X_n) = \begin{cases} 1 & \text{if } \alpha \text{ is true} \\ 0 & \text{otherwise} \end{cases}
\]

Note that the function \( B_\alpha^n \) is determined by both the formula \( \alpha \) and the choice of \( n \). In particular, \( \alpha \) does not need to include all the symbols in \( A_1, \ldots, A_n \).

Formulas and the Boolean Functions they Realize

**Theorem**

Let \( \alpha \) and \( \beta \) be wffs whose sentence symbols are among \( A_1, \ldots, A_n \).

(a) \( \alpha \models \beta \) iff \( B^n_\alpha(\bar{x}) \leq B^n_\beta(\bar{x}) \) for all \( \bar{x} \in \{\text{F}, \text{T}\}^n \).

(b) \( \alpha \) is tautologically equivalent to \( \beta \) iff \( B^n_\alpha = B^n_\beta \).

(c) \( \models \beta \) iff the range of \( B^n_\beta \) is \( \{\text{T}\} \).

**Proof**

(a) \( \alpha \models \beta \) iff every truth assignment satisfying \( \alpha \) also satisfies \( \beta \)

(b) Follows from (a) and \( X = Y \) iff \( X \leq Y \) and \( Y \leq X \).

(c) Follows from definition of tautology.

By shifting our focus from formulas to Boolean functions, tautologically equivalent wffs are identified.
Completeness of Propositional Connectives

**Theorem (Post 1921)**

Let $G$ be an $n$-place Boolean function, $n \geq 1$. There exists a wff $\alpha$ such that $G = B^n_\alpha$, i.e., such that $\alpha$ realizes the function $G$.

**Proof**

If the range of $G$ is just \{F\}, then let $\alpha = A_1 \land \neg A_1$. Clearly, $B^n_{\alpha} = G$.

Otherwise, $G = T$ somewhere. Suppose there are $k$ points where $G = T$:

- $G(X_{11}, X_{12}, \ldots, X_{1n}) = T$
- $G(X_{21}, X_{22}, \ldots, X_{2n}) = T$
- \ldots
- $G(X_{k1}, X_{k2}, \ldots, X_{kn}) = T$

Let $\beta_{ij} = \begin{cases} A_j & \text{if } X_{ij} = T \\ \neg A_j & \text{if } X_{ij} = F \end{cases}$

Then $\alpha$ realizes $G$.

---

**Completeness of Propositional Connectives**

**Example**

Let $G$ be a 3-place Boolean function defined as follows:

- $G(\mathbf{F}, \mathbf{F}, \mathbf{F}) = \mathbf{F}$
- $G(\mathbf{F}, \mathbf{F}, \mathbf{T}) = \mathbf{T}$
- $G(\mathbf{F}, \mathbf{T}, \mathbf{F}) = \mathbf{T}$
- $G(\mathbf{F}, \mathbf{T}, \mathbf{T}) = \mathbf{F}$
- $G(\mathbf{T}, \mathbf{F}, \mathbf{F}) = \mathbf{T}$
- $G(\mathbf{T}, \mathbf{F}, \mathbf{T}) = \mathbf{F}$
- $G(\mathbf{T}, \mathbf{T}, \mathbf{F}) = \mathbf{F}$
- $G(\mathbf{T}, \mathbf{T}, \mathbf{T}) = \mathbf{T}$

There are four points at which $G$ is true, so a DNF formula which realizes $G$ is

$$\neg A_1 \land \neg A_2 \land A_3 \lor (\neg A_1 \land A_2 \land \neg A_3) \lor (A_1 \land \neg A_2 \land \neg A_3) \lor (A_1 \land A_2 \land A_3).$$

Note that another formula which realizes $G$ is $A_1 \leftrightarrow A_2 \leftrightarrow A_3$. Thus, adding additional connectives to a complete set may allow a function to be realized more concisely.

---

**Completeness of Propositional Connectives**

**Proof, continued**

We know that $B^n_{\alpha}(\vec{x}) = v(\alpha)$ where $v(A_i) = X_i$.

Since $\alpha = \gamma_1 \lor \gamma_2 \lor \ldots \lor \gamma_k$, it follows that $B^n_{\alpha}(\vec{x}) = \max(B^n_{\gamma_i}(\vec{x}))$.

But by construction, $B^n_{\gamma_i}(\vec{x}) = T$ if $\vec{x} = (X_{i1}, \ldots, X_{in})$.

Thus $B^n_{\alpha}(\vec{x}) = T$ if $\vec{x}$ is one of the points where $G$ is $T$.

This shows that every Boolean function can be realized by a wff. In fact, every Boolean function can be realized by a wff which uses only the connectives $\{\neg, \land, \lor\}$. We say that this set of connectives is **complete**.

The realizing formula is not unique. The formula built is in so-called **disjunctive normal form** (DNF). A formula is in DNF if it is a disjunction of formulas, each of which is a conjunction of **literals**, where a literal is either a propositional symbol or its negation.

Thus, a corollary is that for every wff, there exists a tautologically equivalent wff in disjunctive normal form.

---

**Completeness of Propositional Connectives**

Recall our definition of some basic Boolean functions:

- $I^n_i = B^n_{A_i}$
- $N = B^n_{A_1}$
- $K = B^n_{A_1 \land A_2}$
- $A = B^n_{A_1 \lor A_2}$

Given that $\{\neg, \land, \lor\}$ is complete, it is not hard to see that any Boolean function can be constructed using only the Boolean functions $I$, $N$, $K$, and $A$.

In fact, we can do better. It turns out that $\{\neg, \land\}$ and $\{\neg, \lor\}$ are complete as well.

**Why?**

- $\alpha \lor \beta \leftrightarrow \neg (\neg \alpha \land \neg \beta)$
- $\alpha \land \beta \leftrightarrow \neg (\neg \alpha \lor \neg \beta)$

Using these identities, the completeness can be easily proved by induction.
Incompleteness of Connectives

To prove that some set of connectives is incomplete, we find a property that is true of all wffs built using those connectives, but that is not true for some Boolean function.

**Example**

\{\land, \rightarrow\} is not complete.

**Proof**

Let \(\alpha\) be a wff which uses only these connectives, and let \(v\) be a truth assignment such that \(v(A_i) = T\) for all \(A_i\). We prove by induction that \(v(\alpha) = T\).

**Base Case**

\(v(A_i) = v(A_i) = T\).

**Inductive Case**

\(v(\beta \land \gamma) = \min(v(\beta), v(\gamma)) = \min(T, T) = T\)

\(v(\beta \rightarrow \gamma) = \max(T - v(\alpha), v(\beta)) = \max(F, T) = T\)

Thus, \(v(\alpha) = T\) for all wffs \(\alpha\) built from \(\{\land, \rightarrow\}\). But \(v(\neg A_1) = F\), so there is no such formula tautologically equivalent to \(\neg A_1\). \(\square\)

---

**Other Propositional Connectives**

For each \(n\), there are \(2^{2^n}\) different \(n\)-place Boolean functions \(B(X_1, \ldots, X_n)\).

**Why?**

There are \(2^n\) different input points and 2 possible output values for each input point. \(2^{2^n}\) is also the number of possible \(n\)-ary propositional connectives.

**0-ary connectives**

There are two 0-place Boolean functions: the constants \(F\) and \(T\). We can construct corresponding 0-ary connectives \(\bot\) and \(\top\) with the meaning that \(v(\bot) = F\) and \(v(\top) = T\) regardless of the truth assignment \(v\).

**Unary connectives**

There are four 1-place functions, but these include the two constant functions mentioned above and the identity function. Thus the only additional connective of interest is negation: \(\neg\).

**Binary connectives**

There are sixteen 2-place Boolean functions. They are cataloged in the following table. Note that the first six correspond to 0-ary and unary connectives.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bot)</td>
<td>constant (F)</td>
<td></td>
</tr>
<tr>
<td>(\top)</td>
<td>constant (T)</td>
<td></td>
</tr>
<tr>
<td>(A)</td>
<td>projection of first argument</td>
<td></td>
</tr>
<tr>
<td>(B)</td>
<td>projection of second argument</td>
<td></td>
</tr>
<tr>
<td>(\neg A)</td>
<td>negation of first argument</td>
<td></td>
</tr>
<tr>
<td>(\neg B)</td>
<td>negation of second argument</td>
<td></td>
</tr>
<tr>
<td>(\land)</td>
<td>(A \land B)</td>
<td>and</td>
</tr>
<tr>
<td>(\lor)</td>
<td>(A \lor B)</td>
<td>or</td>
</tr>
<tr>
<td>(\rightarrow)</td>
<td>(A \rightarrow B)</td>
<td>conditional</td>
</tr>
<tr>
<td>(\leftrightarrow)</td>
<td>(A \leftrightarrow B)</td>
<td>bi-conditional</td>
</tr>
<tr>
<td>(\leftarrow)</td>
<td>(B \rightarrow A)</td>
<td>reverse conditional</td>
</tr>
<tr>
<td>(\oplus)</td>
<td>((A \land \neg B) \lor (\neg A \land B))</td>
<td>exclusive or</td>
</tr>
<tr>
<td>(\downarrow)</td>
<td>(\neg(A \lor B))</td>
<td>nor (or Nicod stroke)</td>
</tr>
<tr>
<td>(\mid)</td>
<td>(\neg(A \land B))</td>
<td>nand (or Sheffer stroke)</td>
</tr>
<tr>
<td>(&lt;)</td>
<td>(\neg A \land B)</td>
<td>less than</td>
</tr>
<tr>
<td>(&gt;)</td>
<td>(A \land \neg B)</td>
<td>greater than</td>
</tr>
</tbody>
</table>

**Compactness**

Recall that a wff \(\alpha\) is satisfiable if there exists a truth assignment \(v\) such that \(v(\alpha) = T\).

A set \(\Sigma\) of wffs is satisfiable if there exists a truth assignment \(v\) such that \(v(\alpha) = T\) for each \(\alpha \in \Sigma\).

A set \(\Sigma\) is finitely satisfiable iff every finite subset of \(\Sigma\) is satisfiable.

**Compactness Theorem**

A set of wffs is satisfiable iff it is finitely satisfiable.

**Proof**

The only if direction is trivial since any subset of a satisfiable set is clearly satisfiable.

To prove the other direction, assume that \(\Sigma\) is a set which is finitely satisfiable. We must show that \(\Sigma\) is satisfiable.
Compactness

Let $\Sigma$ be finitely satisfiable. We extend $\Sigma$ to form a *maximal* finitely satisfiable set $\Delta$ as follows.

Let $\alpha_1, \ldots, \alpha_n, \ldots$ be a fixed enumeration of all *wffs*.

Why is this possible? The set of all sequences of a countable set is countable.

Then, let

$\Delta_0 = \Sigma,$

$\Delta_{n+1} = \begin{cases} 
\Delta_n \cup \{\alpha_{n+1}\} & \text{if this is finitely satisfiable,} \\
\Delta_n \cup \{\neg \alpha_{n+1}\} & \text{otherwise.}
\end{cases}$

It is not hard to show that each $\Delta_n$ is finitely satisfiable.

Let $\Delta = \bigcup_n \Delta_n$. It is then clear that

1. $\Sigma \subseteq \Delta$
2. $\alpha \in \Delta$ or $\neg \alpha \in \Delta$ for any *wff* $\alpha$, and
3. $\Delta$ is finitely satisfiable.

---

Compactness

Now we show that $\Delta$ is satisfiable (and thus $\Sigma \subseteq \Delta$ is also satisfiable).

Define a truth assignment $v$ as follows. For each propositional symbol $A_i$,

$v(A_i) = T$ iff $A_i \in \Delta$.

We claim that for any *wff* $\alpha$, $v$ satisfies $\alpha$ iff $\alpha \in \Delta$. The proof is by induction on well-formed formulas.

**Base Case**

Follows directly from the definition of $v$.

**Induction Case**

We will just consider one case. Suppose $\alpha = \beta \land \gamma$. Then

$v(\alpha) = T$ iff both $v(\beta) = T$ and $v(\gamma) = T$ if both $\beta \in \Delta$ and $\gamma \in \Delta$.

Now, if both $\beta$ and $\gamma$ are in $\Delta$, then since $\{\beta, \gamma, \neg \alpha\}$ is not satisfiable, we must have $\alpha \notin \Delta$.

Similarly, if one of $\beta$ or $\gamma$ is not in $\Delta$, then its negation must be in $\Delta$, so $\alpha \notin \Delta$. $\square$

---

**Corollary**

If $\Sigma \models \alpha$ then there is a finite $\Sigma_0 \subseteq \Sigma$ such that $\Sigma_0 \models \alpha$.

**Proof**

Suppose that $\Sigma_0 \not\models \alpha$ for every finite $\Sigma_0 \subseteq \Sigma$.

Then, $\Sigma_0 \cup \{\neg \alpha\}$ is satisfiable for every finite $\Sigma_0 \subseteq \Sigma$.

So, by compactness, $\Sigma \cup \{\neg \alpha\}$ is satisfiable which contradicts the fact that $\Sigma \models \alpha$.

$\square$