VL LOGIK: PROPOSITIONAL LOGIC

WS 2016/2017 (342.208)

Armin Biere, FMV (biere@jku.at)
Martina Seidl, FMV (martina.seidl@jku.at)
Version 2016.3
Propositions

A proposition is an atomic statement that is either true or false.

Example:
- Alice comes to the party.
- It rains.

With connectives, propositions can be combined.

Example:
- Alice comes to the party, Bob as well, but not Cecile.
- If it rains, the street is wet.
Propositional Logic

- **two truth values (Boolean domain):** true/false, verum/falsum, on/off, 1/0

- **language elements**
  - atomic propositions (atoms, variables)
    - no internal structure
    - either true or false
  - logic connectives: not (¬), and (∧), or (∨), ... 
    - operators for construction of composite propositions
    - concise meaning
    - argument(s) and return value from Boolean domain
  - parenthesis

**Example:** formula of propositional logic: \((\neg t \lor s) \land (t \lor s) \land (\neg t \lor \neg s)\)

Atoms: \(t, s\), connectives: \(\neg, \lor, \land\), parenthesis for structuring the expression
Background

- *historical origins*: ancient Greeks
- in philosophy, mathematics, and computer science
- two very basic principles:
  - Law of Excluded Middle:
    A proposition is true or its negation is true.
  - Law of Contradiction:
    No expression is both true and false at the same time.
- very *simple* language
  - no objects, no arguments to propositions
  - no functions, no quantifiers
- solving is *easy* (relative to other logics)
- many applications in industry
Syntax of Propositional Logic (1/2)

The set $\mathcal{L}$ of well-formed propositional formulas is the smallest set such that

1. $\top, \bot \in \mathcal{L}$;
2. $\mathcal{P} \subseteq \mathcal{L}$ where $\mathcal{P}$ is the set of atomic propositions (atoms, variables);
3. if $\phi \in \mathcal{L}$ then $(\neg \phi) \in \mathcal{L}$;
4. if $\phi, \psi \in \mathcal{L}$ then $(\phi \circ \psi) \in \mathcal{L}$ with $\circ \in \{\lor, \land, \leftrightarrow, \rightarrow\}$.

$\mathcal{L}$ is the language of propositional logic. The elements of $\mathcal{L}$ are *propositional formulas*. 
Syntax of Propositional Logic (2/2)

In Backus-Naur form (BNF) propositional formulas are described as follows:

\[ \phi ::= \top | \bot | p | (\neg \phi) | (\phi \lor \phi) | (\phi \land \phi) | (\phi \leftrightarrow \phi) | (\phi \rightarrow \phi) \]

Example:

- \( \top \)
- \( \neg a \)
- \( \neg(\neg a) \)
- \( \neg(a \lor b) \)
- \( a \)
- \( \neg \bot \)
- \( a_1 \lor a_2 \)
- \( \neg(a \leftrightarrow b) \)
- \( (((\neg a) \lor a') \leftrightarrow (b \rightarrow c)) \)
- \( (((a_1 \lor a_2) \lor (a_3 \land \bot)) \rightarrow b) \)
Rules of Precedence

To reduce the number of parenthesis, we use the following conventions (in case of doubt, uses parenthesis!):

- \( \neg \) is stronger than \( \land \)
- \( \land \) is stronger than \( \lor \)
- \( \lor \) is stronger than \( \rightarrow \)
- \( \rightarrow \) is stronger than \( \leftrightarrow \)
- Binary operators of same strength are assumed to be left parenthesized (also called “left associative”)

Example:

- \( \neg a \land b \lor c \rightarrow d \leftrightarrow f \) is the same as (((((\neg a) \land b) \lor c) \rightarrow d) \leftrightarrow f).
- \( a' \lor a'' \lor a'' \land b' \lor b'' \) is the same as (((a' \lor a'') \lor (a'' \land b')) \lor b'').
- \( a' \land a'' \land a''' \lor b' \land b'' \) is the same as (((a' \land a'') \land a''') \lor (b' \land b'')).
Formula Tree

- formulas have a tree structure
  - *inner nodes*: connectives
  - *leaves*: truth constants, variables

- *default*: inner nodes have **one** child node (negation) or **two** nodes as children (other connectives).

- tree structure reflects the use of parenthesis

- *simplification*:
  - disjunction and conjunction may be considered as \( n \)-ary operators,
  - i.e., if a node \( N \) and its child node \( C \) are of the same kind of connective (conjunction / disjunction), then the children of \( C \) can become direct children of \( N \) and the \( C \) is removed.
The formula

\[(a \lor (b \lor \lnot c)) \leftrightarrow (\top \land ((a \rightarrow \lnot b) \lor (\bot \lor a \lor b)))\]

has the formula tree

\[
\begin{align*}
\leftrightarrow & \quad \lor \\
\lor & \quad \land \\
a & \quad \lor \\
b & \quad \neg \\
c & \\
\rightarrow & \quad \lor \\
a & \quad \lor \\
\bot & \\
\top & \\
b & \\
\land & \\
b & \quad \lor \\
\bot & \quad a
\end{align*}
\]
The formula

\[(a \lor (b \lor \neg c)) \iff (\top \land ((a \rightarrow \neg b) \lor (\bot \lor a \lor b)))\]

has the simplified formula tree
Subformulas

An *immediate subformula* is defined as follows:

- truth constants and atoms have no immediate subformula.
- only immediate subformula of $\neg \phi$ is $\phi$.
- formula $\phi \circ \psi$ ($\circ \in \{\land, \lor, \leftrightarrow, \rightarrow\}$) has immediate subformulas $\phi$ and $\psi$.

*Informal*: a subformula is a formula that is part of a formula

The *set of subformulas* of a formula $\phi$ is the smallest set $S$ with

1. $\phi \in S$
2. if $\psi \in S$ then all immediate subformulas of $\psi$ are in $S$

The subformulas of $(a \lor b) \rightarrow (c \land \neg \neg d)$ are

$$\{a, b, c, d, \neg d, \neg \neg d, a\lor b, c\land \neg \neg d, (a\lor b) \rightarrow (c\land \neg \neg d)\}$$
Limboole

- SAT-solver
- available at http://fmv.jku.at/limboole/
- input format in BNF:

\[
\begin{align*}
\langle expr \rangle & ::= \langle iff \rangle \\
\langle iff \rangle & ::= \langle implies \rangle \mid \langle implies \rangle "\leftrightarrow" \langle implies \rangle \\
\langle implies \rangle & ::= \langle or \rangle \mid \langle or \rangle "\rightarrow" \langle or \rangle \mid \langle or \rangle "\leftarrow" \langle or \rangle \\
\langle or \rangle & ::= \langle and \rangle \mid \langle and \rangle "\mid" \langle and \rangle \\
\langle and \rangle & ::= \langle not \rangle \mid \langle not \rangle "\&" \langle not \rangle \\
\langle not \rangle & ::= \langle basic \rangle \mid "!" \langle not \rangle \\
\langle basic \rangle & ::= \langle var \rangle \mid "(" \langle expr \rangle ")"
\end{align*}
\]

where 'var' is a string over letters, digits, and \(- _ . [ ] $ @\)

In Limboole the formula \((a \lor b) \rightarrow (c \land \neg \neg d)\) is represented as

\[
((a \mid b) \rightarrow (c \& \neg \neg d))
\]
Special Formula Structures

- **literal**: variable or a negated variable (also (negated) truth constants)
  - examples of literals: $x, \neg x, y, \neg y$
  - If $l$ is a literal with $l = x$ or $l = \neg x$ then $\text{var}(l) = x$.
  - For literals we use letter $l, k$ (possibly indexed or primed).
  - In principle, we identify $\neg \neg l$ with $l$.

- **clause**: disjunction of literals
  - unary clause (clause of size one): $l$ where $l$ is a literal
  - empty clause (clause of size zero): $\bot$
  - examples of clauses: $(x \lor y), (\neg x \lor x' \lor \neg x''), x, \neg y$

- **cube**: conjunction of literals
  - unary cube (cubes of size one): $l$ where $l$ is a literal
  - empty cubes (cubes of size zero): $\top$
  - examples of cubes: $(x \land y), (\neg x \land x' \land \neg x'')$, $x, \neg y$
Negation Normal Form (1/2)

**Negation Normal Form (NNF)** is defined as follows:

- Literals and truth constants are in NNF;
- $\phi \circ \psi$ ($\circ \in \{\vee, \wedge\}$) is in NNF iff $\phi$ and $\psi$ are in NNF;
- no other formulas are in NNF.

In other words: A formula in NNF contains only conjunctions, disjunctions, and negations and negations only occur in front of variables and constants.
Negation Normal Form (2/2)

If a formula is in negation normal form then

- in the formula tree, nodes with negation symbols only occur directly before leaves.
- there are no subformulas of the form $\neg \phi$ where $\phi$ is something else than a variable or a constant.
- it does not contain NAND, NOR, XOR, equivalence, and implication connectives.

**Example:** The formula $((x \lor \neg x_1) \land (x \lor (z \lor \neg x_1)))$ is in NNF but $\neg((x \lor \neg x_1) \land (x \lor (z \lor \neg x_1)))$ is not in NNF.
Conjunctive Normal Form (CNF)

A propositional formula is in *conjunctive normal form* (CNF) iff it is a conjunction of clauses.

A formula in conjunctive normal form is

- in negation normal form
- \( \top \) if it contains no clauses
- easy to check whether it can be refuted

*remark*: CNF is the input of most SAT-solvers (DIMACS format)
Disjunctive Normal Form (DNF)

A propositional formula is in *disjunctive normal form* (DNF) if it is a disjunction of cubes.

A formula in disjunctive normal form is

- in negation normal form
- \( \bot \) if it contains no cubes
- easy to check whether it can be satisfied
Examples for CNF and DNF

Examples CNF

- $\top$
  - $l_1 \land l_2 \land l_3$
- $\bot$
  - $l_1 \lor l_2 \lor l_3$
- $a$
  - $(a_1 \lor \neg a_2) \land (a_1 \lor b_2 \lor a_2) \land a_2$
- $\neg a$
  - $((l_{11} \lor \ldots \lor l_{1m_1}) \land \ldots \land (l_{n1} \lor \ldots \lor l_{nm_n}))$

Examples DNF

- $\top$
  - $l_1 \land l_2 \land l_3$
- $\bot$
  - $l_1 \lor l_2 \lor l_3$
- $a$
  - $(a_1 \land \neg a_2) \lor (a_1 \land b_2 \land a_2) \lor a_2$
- $\neg a$
  - $((l_{11} \land \ldots \land l_{1m_1}) \lor \ldots \lor (l_{n1} \land \ldots \land l_{nm_n}))$
we use the following conventions unless stated otherwise:

- $a, b, c, x, y, z$ denote variables and $l, k$ denote literals
- $\phi, \psi, \gamma$ denote arbitrary formulas
- $C, D$ denote clauses or cubes (clear from context)
- clauses are also written as sets
  - $(l_1 \lor \ldots \lor l_n) = \{l_1, \ldots l_n\}$
  - to add a literal $l$ to clause $C$, we write $C \cup \{l\}$
  - to remove a literal $l$ from clause $C$, we write $C \setminus \{l\}$
- formulas in CNF are also written as sets of sets
  - $((l_{11} \lor \ldots \lor l_{1m_1}) \land \ldots \land (l_{n1} \lor \ldots \lor l_{nm_n})) = \\{\{l_{11}, \ldots l_{1m_1}\}, \ldots, \{l_{n1}, \ldots l_{nm_n}\}\}$
  - to add a clause $C$ to CNF $\phi$, we write $\phi \cup \{C\}$
  - to remove a clause $C$ from CNF $\phi$, we write $\phi \setminus \{C\}$
Negation

- unary connective $\neg$ (operator with exactly one argument)
- negating the truth value of its argument
- alternative notation: $!\phi$, $\bar{\phi}$, $\neg\phi$, $\text{NOT}\phi$

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\neg\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Example:
- If the atom “It rains.” is true then the negation “It does not rain.” is false.
- If atom $a$ is true then $\neg a$ is false.
- If formula $((a \lor x) \land y)$ is true then formula $\neg((a \lor x) \land y)$ is false.
- If formula $((b \rightarrow y) \land z)$ is true then formula $\neg((b \rightarrow y) \land z)$ is false.
Conjunction

- A conjunction is true iff both arguments are true.
- Alternative notation for $\phi \land \psi$: $\phi \& \psi$, $\phi \psi$, $\phi \ast \psi$, $\phi \cdot \psi$, $\phi \text{AND} \psi$

For $(\phi_1 \land \ldots \land \phi_n)$ we also write $\bigwedge_{i=1}^{n} \phi_i$.

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\psi$</th>
<th>$\phi \land \psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table:

Set view:

**Example:**

- $(a \land \neg a)$ is always false.
- $(\top \land a)$ is true if $a$ is true. $(\bot \land \phi)$ is always false.
- If $(a \lor b)$ is true and $(\neg c \lor d)$ is true then $(a \lor b) \land (\neg c \lor d)$ is true.
Disjunction

- a disjunction is true iff at least one of the arguments is true
- alternative notation for $\phi \lor \psi$: $\phi | \psi$, $\phi + \psi$, $\phi OR \psi$
- For $(\phi_1 \lor \ldots \lor \phi_n)$ we also write $\lor_{i=1}^{n} \phi_i$.

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\psi$</th>
<th>$\phi \lor \psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Example:

- $(a \lor \neg a)$ is always true.
- $(\top \lor a)$ is always true. $(\bot \lor a)$ is true if $a$ is true.
- If $(a \rightarrow b)$ is true and $(\neg c \rightarrow d)$ then $(a \rightarrow b) \lor (\neg c \rightarrow d)$ is true.
Implication

- an implication is true iff the first argument is false or both arguments are true (Ex falsum quodlibet.)
- alternative notation: \( \phi \supset \psi, \phi \text{ IMPL } \psi \)

**Truth table:**

<table>
<thead>
<tr>
<th>( \phi )</th>
<th>( \psi )</th>
<th>( \phi \rightarrow \psi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Set view:**

Example:

- If atom "It rains." is true and atom "The street is wet." is true then the statement "If it rains, the street is wet." is true.
- \( (\bot \rightarrow a) \) and \( (a \rightarrow a) \) are always true. \( \top \rightarrow \phi \) is true if \( \phi \) is true.
Equivalence

- true iff both subformulas have the same value
- alternative notation: $\phi = \psi, \phi \equiv \psi, \phi \sim \psi$

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\psi$</th>
<th>$\phi \leftrightarrow \psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

truth table:

Example:

- The formula $a \leftrightarrow a$ is always true.
- The formula $a \leftrightarrow b$ is true iff $a$ is true and $b$ is true or $a$ is false and $b$ is false.
- $\top \leftrightarrow \bot$ is never true.
The Logic Connectives at a Glance

<table>
<thead>
<tr>
<th>φ</th>
<th>ψ</th>
<th>T</th>
<th>⊥</th>
<th>¬φ</th>
<th>φ ∧ ψ</th>
<th>φ ∨ ψ</th>
<th>φ → ψ</th>
<th>φ ↔ ψ</th>
<th>φ ⊕ ψ</th>
<th>φ ↑ ψ</th>
<th>φ ↓ ψ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Example:**

<table>
<thead>
<tr>
<th>φ</th>
<th>ψ</th>
<th>¬(¬φ ∧ ¬ψ)</th>
<th>¬φ ∨ ψ</th>
<th>(φ → ψ) ∧ (ψ → φ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Observation:** connectives can be expressed by other connectives.
Other Connectives

- there are 16 different functions for binary connectives
- so far, we had $\land, \lor, \leftrightarrow, \rightarrow$
- further connectives:
  - $\phi \not\leftrightarrow \psi$ (also $\oplus$, xor, antivalence)
  - $\phi \uparrow \psi$ (nand, Sheffer Stroke Function)
  - $\phi \downarrow \psi$ (nor, Pierce Function)

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\psi$</th>
<th>$\phi \not\leftrightarrow \psi$</th>
<th>$\phi \uparrow \psi$</th>
<th>$\phi \downarrow \psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- nor and nand can express every other boolean function (i.e., they are functional complete)
- often used for building digital circuits (like processors)
Propositional Formulas and Digital Circuits

- **and gate**
  - $A$ & $B$

- **nand gate**
  - $A$ & $B$

- **or gate**
  - $A$ & $B$

- **nor gate**
  - $A$ & $B$

- **xor gate**
  - $A$ & $B$

- **not gate**
  - $A$
Example of a Digital Circuit: Half Adder

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$c$</th>
<th>$s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

From the truth table, we see that:

$$c \iff x \land y$$

and

$$s \iff x \oplus y.$$
### Different Notations

<table>
<thead>
<tr>
<th>operator</th>
<th>logic</th>
<th>circuits</th>
<th>C/C++/Java/C#</th>
<th>VHDL</th>
<th>Limboole</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\top$</td>
<td>1</td>
<td>true</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td>0</td>
<td>$\bot$</td>
<td>0</td>
<td>false</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>negation</td>
<td>$\neg \phi$</td>
<td>$\bar{\phi}$</td>
<td>$\neg \phi$</td>
<td>not $\phi$</td>
<td>$! \phi$</td>
</tr>
<tr>
<td>conjunction</td>
<td>$\phi \land \psi$</td>
<td>$\phi \psi$</td>
<td>$\phi \cdot \psi$</td>
<td>$\phi &amp;&amp; \psi$</td>
<td>$\phi \text{ and } \psi$</td>
</tr>
<tr>
<td>disjunction</td>
<td>$\phi \lor \psi$</td>
<td>$\phi + \psi$</td>
<td>$\phi</td>
<td></td>
<td>\psi$</td>
</tr>
<tr>
<td>exclusive or</td>
<td>$\phi \leftrightarrow \psi$</td>
<td>$\phi \oplus \psi$</td>
<td>$\phi != \psi$</td>
<td>$\phi \text{xor } \psi$</td>
<td>–</td>
</tr>
<tr>
<td>implication</td>
<td>$\phi \rightarrow \psi$</td>
<td>$\phi \supset \psi$</td>
<td>–</td>
<td>–</td>
<td>$\phi \rightarrow \psi$</td>
</tr>
<tr>
<td>equivalence</td>
<td>$\phi \leftrightarrow \psi$</td>
<td>$\phi = \psi$</td>
<td>$\phi == \psi$</td>
<td>$\phi \text{ xnor } \psi$</td>
<td>$\phi \leftrightarrow \psi$</td>
</tr>
</tbody>
</table>

**Example:**

- $(a \lor (b \lor \neg c)) \leftrightarrow (\top \land ((a \rightarrow \neg b) \lor (c \lor a \lor b)))$
- $(a + (b + \bar{c})) = c \ ((a \supset \neg b) + (0 + a + b))$
- $(a || (b || !c)) == (c \&\& ((! a || ! b) || (false || a || b)))$
# All 16 Binary Functions

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\psi$</th>
<th>constant 0</th>
<th>nor</th>
<th>xor</th>
<th>nand</th>
<th>and</th>
<th>equivalence</th>
<th>implication</th>
<th>or</th>
<th>constant 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 1 1 0 1 0 1 0</td>
<td>1 0 1 0 1</td>
<td>0 1 0</td>
<td>1 0 1 0 1</td>
<td>0 1 0 1 0</td>
<td>1 0 1 0 1</td>
<td>0 1 1 0 1</td>
<td>0 1 1 0 1</td>
<td>0 1 1 0 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0 0 1 1 0 0 1 1</td>
<td>0 1 1 0 0</td>
<td>0 1 1</td>
<td>0 1 1 0 0</td>
<td>1 0 1 1 0</td>
<td>0 0 1 1 0</td>
<td>0 0 1 1 0</td>
<td>0 0 1 1 0</td>
<td>0 0 1 1 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0 0 0 0 0 1 1 1</td>
<td>1 1 1 1 0</td>
<td>0 0 0</td>
<td>0 1 1 0 0</td>
<td>0 0 0 0 0</td>
<td>1 1 1 1 0</td>
<td>0 0 0 0 0</td>
<td>1 1 1 0 0</td>
<td>1 1 1 1 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0</td>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1</td>
</tr>
</tbody>
</table>
Assignment

■ A variable can be assigned one of two values from the two-valued domain $\mathbb{B}$, where $\mathbb{B} = \{1, 0\}$

■ The mapping $\nu : \mathcal{P} \rightarrow \mathbb{B}$ is called assignment, where $\mathcal{P}$ is the set of atomic propositions

■ We sometimes write an assignment $\nu$ as set $V$ with $V \subseteq \mathcal{P} \cup \{\neg x | x \in \mathcal{P}\}$ such that
  - $x \in V$ iff $\nu(x) = 1$
  - $\neg x \in V$ iff $\nu(x) = 0$

■ For $n$ variables, there are $2^n$ assignments possible

■ An assignment corresponds to one line in the truth table
Assignment: Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>(x ∨ y) ∧ ¬z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- one assignment: \( \nu(x) = 1, \nu(y) = 0, \nu(z) = 1 \)
- alternative notation: \( V = \{ x, ¬y, z \} \)
- observation: A variable assignment determines the truth value of the formulas containing these variables.
Semantics of Propositional Logic

Given assignment $\nu : \mathcal{P} \rightarrow \mathbb{B}$, the interpretation $[.]_\nu : \mathcal{L} \rightarrow \mathbb{B}$ is defined by:

- $[\top]_\nu = 1$, $[\bot]_\nu = 0$
- if $x \in \mathcal{P}$ then $[x]_\nu = \nu(x)$
- $[\neg \phi]_\nu = 1$ iff $[\phi]_\nu = 0$
- $[\phi \lor \psi]_\nu = 1$ iff $[\phi]_\nu = 1$ or $[\psi]_\nu = 1$
Satisfying/Falsifying Assignments

■ An assignment is called
  □ **satisfying** a formula $\phi$ iff $[\phi]_\nu = 1$.
  □ **falsifying** a formula $\phi$ iff $[\phi]_\nu = 0$.

■ A satisfying assignment for $\phi$ is a **model** of $\phi$.
■ A falsifying assignment for $\phi$ is a **counter-model** of $\phi$.

**Example:**

For formula $((x \lor y) \land \neg z)$,
■ $\{x, y, z\}$ is a counter-model,
■ $\{x, y, \neg z\}$ is a model.
Properties of Propositional Formulas (1/3)

- formula $\phi$ is **satisfiable** iff there exists interpretation $[.]_\nu$ with $[\phi]_\nu = 1$
  
  check with limboole -s

- formula $\phi$ is **valid** iff for all interpretations $[.]_\nu$ it holds that $[\phi]_\nu = 1$
  
  check with limboole

- formula $\phi$ is **refutable** iff exists interpretation $[.]_\nu$ with $[\phi]_\nu = 0$
  
  check with limboole

- formula $\phi$ is **unsatisfiable** iff $[\phi]_\nu = 0$ for all interpretations $[.]_\nu$
  
  check with limboole -s
Properties of Propositional Formulas (2/3)

- A valid formula is called \textit{tautology}.
- An unsatisfiable formula is called \textit{contradiction}.

Example:

- \( \top \) is valid.
- \( \bot \) is unsatisfiable.
- \( (a \lor \neg b) \land (\neg a \lor b) \) is refutable.
- \( a \rightarrow b \) is satisfiable.
- \( a \leftrightarrow \neg a \) is a contradiction.
- \( (a \lor \neg b) \land (\neg a \lor b) \) is satisfiable.
Properties of Propositional Formulas (3/3)

- A satisfiable formula is
  - possibly valid
  - possibly refutable
  - not unsatisfiable.

- A valid formula is
  - satisfiable
  - not refutable
  - not unsatisfiable.

- A refutable formula is
  - possibly satisfiable
  - possibly unsatisfiable
  - not valid.

- An unsatisfiable formula is
  - refutable
  - not valid
  - not satisfiable.

Example:
- satisfiable, but not valid: \( a \leftrightarrow b \)
- satisfiable and refutable: \( (a \lor b) \land (\neg a \lor c) \)
- valid, not refutable \( \top \lor (a \land \neg a) \); not valid, refutable \( (\bot \lor b) \)
Further Connections between Formulas

■ A formula $\phi$ is valid iff $\neg \phi$ is unsatisfiable.

■ A formula $\phi$ is satisfiable iff $\neg \phi$ is not valid.

■ The formulas $\phi$ and $\psi$ are equivalent iff $\phi \leftrightarrow \psi$ is valid.

■ The formulas $\phi$ and $\psi$ are equivalent iff $\neg (\phi \leftrightarrow \psi)$ is unsatisfiable.

■ A formula $\phi$ is satisfiable iff $\phi \not\leftrightarrow \bot$. 
Simple Algorithm for Satisfiability Checking

1. **Algorithm:** evaluate
   
   **Data:** formula $\phi$
   
   **Result:** 1 iff $\phi$ is satisfiable

2. if $\phi$ contains a variable $x$ then
   
   pick $v \in \{\top, \bot\}$
   
   /* replace $x$ by truth constant $v$, evaluate resulting formula */

3. if evaluate($\phi[x|v]$) then return 1;

4. else return evaluate($\phi[x|\overline{v}]$);

else

5. switch $\phi$ do

6. case $\top$ do return 1;

7. case $\bot$ do return 0;

8. case $\neg \psi$ do return ! evaluate($\psi$) /* true iff $\psi$ is false */;

9. case $\psi' \land \psi''$ do

10. return evaluate($\psi'$) && evaluate($\psi''$) /* true iff both $\psi'$ and $\psi''$ are true */;

11. case $\psi' \lor \psi''$ do

12. return evaluate($\psi'$) || evaluate($\psi''$) /* true iff $\psi'$ or $\psi''$ is true */;
Semantic Equivalence

Two formula $\phi$ and $\psi$ are \textit{semantic equivalent} (written as $\phi \iff \psi$) iff for all interpretations $[.]_\nu$ it holds that $[\phi]_\nu = [\psi]_\nu$.

- $\iff$ is a \textit{meta-symbol}, i.e., it is not part of the language.
- \textit{natural language}: if and only if (iff)
- $\phi \iff \psi$ iff $\phi \leftrightarrow \psi$ is valid, i.e., we can express semantics by means of syntactics.
- If $\phi$ and $\psi$ are not equivalent, we write $\phi \nleftrightarrow \psi$.

\textbf{Example:}

- $a \lor \neg a \nleftrightarrow b \rightarrow \neg b$
- $(a \lor b) \land \neg (a \lor b) \iff \bot$
- $a \lor \neg a \iff b \lor \neg b$
- $a \leftrightarrow (b \leftrightarrow c)) \iff ((a \leftrightarrow b) \leftrightarrow c$
### Examples of Semantic Equivalences (1/2)

<table>
<thead>
<tr>
<th>Equation</th>
<th>Equation</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi \land \psi \leftrightarrow \psi \land \phi )</td>
<td>( \phi \lor \psi \leftrightarrow \psi \lor \phi )</td>
<td>commutativity</td>
</tr>
<tr>
<td>( \phi \land (\psi \land \gamma) \leftrightarrow (\phi \land \psi) \land \gamma )</td>
<td>( \phi \lor (\psi \lor \gamma) \leftrightarrow (\phi \lor \psi) \lor \gamma )</td>
<td>associativity</td>
</tr>
<tr>
<td>( \phi \land (\phi \lor \psi) \leftrightarrow \phi )</td>
<td>( \phi \lor (\phi \land \psi) \leftrightarrow \phi )</td>
<td>absorption</td>
</tr>
<tr>
<td>( \phi \land (\psi \lor \gamma) \leftrightarrow (\phi \land \psi) \lor (\phi \land \gamma) )</td>
<td>( \phi \lor (\psi \land \gamma) \leftrightarrow (\phi \lor \psi) \land (\phi \lor \gamma) )</td>
<td>distributivity</td>
</tr>
<tr>
<td>( \neg (\phi \land \psi) \leftrightarrow \neg \phi \lor \neg \psi )</td>
<td>( \neg (\phi \lor \psi) \leftrightarrow \neg \phi \land \neg \psi )</td>
<td>laws of De Morgan</td>
</tr>
<tr>
<td>( \phi \leftrightarrow \psi \leftrightarrow (\phi \rightarrow \psi) \land (\psi \rightarrow \phi) )</td>
<td>( \phi \leftrightarrow \psi \leftrightarrow (\phi \land \psi) \lor (\neg \phi \land \neg \psi) )</td>
<td>synt. equivalence</td>
</tr>
</tbody>
</table>
Examples of Semantic Equivalences (2/2)

| $\phi \lor \psi \iff \neg\phi \rightarrow \psi$ | $\phi \rightarrow \psi \iff \neg\psi \rightarrow \neg\phi$ | implications |
| $\phi \land \neg\phi \iff \bot$ | $\phi \lor \neg\phi \iff \top$ | complement |
| $\neg\neg\phi \iff \phi$ | | double negation |
| $\phi \land \top \iff \phi$ | $\phi \lor \bot \iff \phi$ | neutrality |
| $\phi \lor \top \iff \top$ | $\phi \land \bot \iff \bot$ | |
| $\neg\top \iff \bot$ | $\neg\bot \iff \top$ | |

| $\neg\neg\phi \iff \phi$ | | double negation |
| $\phi \land \top \iff \phi$ | $\phi \lor \bot \iff \phi$ | neutrality |
| $\phi \lor \top \iff \top$ | $\phi \land \bot \iff \bot$ | |
| $\neg\top \iff \bot$ | $\neg\bot \iff \top$ | |

| $\neg\neg\phi \iff \phi$ | | double negation |
| $\phi \land \top \iff \phi$ | $\phi \lor \bot \iff \phi$ | neutrality |
| $\phi \lor \top \iff \top$ | $\phi \land \bot \iff \bot$ | |
| $\neg\top \iff \bot$ | $\neg\bot \iff \top$ | |
Logic Entailment

Let $\phi_1, \ldots, \phi_n, \psi$ be propositional formulas. Then $\phi_1, \ldots, \phi_n$ entail $\psi$ (written as $\phi_1, \ldots, \phi_n \models \psi$) iff $[\phi_1]_\nu = 1, \ldots, [\phi_n]_\nu = 1$ implies that $[\psi]_\nu = 1$.

Informal meaning: True premises derive a true conclusion.

- $\models$ is a meta-symbol, i.e., it is not part of the language.
- $\phi_1, \ldots, \phi_n \models \psi$ iff $(\phi_1 \land \ldots \land \phi_n) \rightarrow \psi$ is valid, i.e., we can express semantics by means of syntactics.
- If $\phi_1, \ldots, \phi_n$ do not entail $\psi$, we write $\phi_1, \ldots, \phi_n \not\models \psi$.

Example:

- $a \models a \lor b$
- $\models a \lor \neg a$
- $a, a \rightarrow b \models b$
- $a, b \models a \land b$
- $\not\models a \land \neg a$
- $\bot \models a \land \neg a$
Satisfiability Equivalence

Two formulas $\phi$ and $\psi$ are *satisfiability-equivalent* (written as $\phi \iff_{SAT} \psi$) iff both formulas are satisfiable or both are contradictory.

- Satisfiability-equivalent formulas are not necessarily satisfied by the same assignments.
- Satisfiability equivalence is a weaker property than semantic equivalence.
- Often sufficient for simplification rules: If the complicated formula is satisfiable then also the simplified formula is satisfiable.
**Example: Satisfiability Equivalence**

*positive pure literal elimination rule:*

If a variable $x$ occurs in a formula but $\neg x$ does not occur in the formula, then $x$ can be substituted by $\top$. The resulting formula is satisfiability-equivalent.

---

**Example:**

- $x \Leftrightarrow_{SAT} \top$, but $x \nLeftrightarrow \top$
- $(a \land b) \lor (\neg c \land a) \Leftrightarrow_{SAT} b \lor \neg c$, but $(a \land b) \lor (\neg c \land a) \nLeftrightarrow b \lor \neg c$
Representing Functions as CNFs

Problem: Given the truth table of a Boolean function $\phi$. How is the function represented in propositional logic?

Solution (in CNF):

1. Represent each assignment $\nu$ where $\phi$ has value 0 as clause:
   - If variable $x$ is 1 in $\nu$, add $\neg x$ to clause.
   - If variable $x$ is 0 in $\nu$, add $x$ to clause.

2. Connect all clauses by conjunction.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>$\phi$</th>
<th>clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$a \lor b \lor c$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$a \lor \neg b \lor \neg c$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$\neg a \lor b \lor \neg c$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$\neg a \lor \neg b \lor c$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$\phi = (a \lor b \lor c) \land (a \lor \neg b \lor \neg c) \land (\neg a \lor \neg b \lor c)$
Representing Functions as DNFs

**Problem:** Given the truth table of a Boolean function $\phi$. How is the function represented in propositional logic?

**Solution (in DNF):**

1. Represent each assignment $\nu$ where $\phi$ has value 1 as cube:
   - If variable $x$ is 1 in $\nu$, add $x$ to cube.
   - If variable $x$ is 0 in $\nu$, add $\neg x$ to cube.

2. Connect all cubes by disjunction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>$\phi$</th>
<th>cubes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\neg a \land \neg b \land c$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$\neg a \land b \land \neg c$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$a \land \neg b \land \neg c$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$a \land b \land c$</td>
</tr>
</tbody>
</table>

$$
\phi = \\
(\neg a \land \neg b \land c) \lor (\neg a \land b \land \neg c) \lor \\
(a \land \neg b \land \neg c) \lor (a \land b \land c)
$$
Functional Completeness

- In propositional logic there are
  - 2 functions of arity 0 ($\top, \bot$)
  - 4 functions of arity 1 (e.g., not)
  - 16 functions of arity 2 (e.g., and, or, ...)
  - $2^n$ functions of arity $n$.
- A function of arity $n$ has $2^n$ different combinations of arguments (lines in the truth table).
- A function maps its arguments either to 1 or 0.

A set of functions is called *functional complete* for propositional logic iff it is possible to express all other functions of propositional logic with functions from this set.

\{\neg, \land\}, \{\neg, \lor\}, \{\text{nand}\} are functional complete.
Encoding the k-Coloring Problem

Given graph \((V, E)\) with vertices \(V\) and edges \(E\). Color each node with one of \(k\) colors, such that there is no edge \((v, w) \in E\), with vertices \(v\) and \(w\) colored in the same color.

Encoding:

1. **Propositional variables**: \(v_j \ldots\) node \(v \in V\) has color \(j\) \((1 \leq j \leq k)\)

2. each node has a color:

\[
\bigwedge_{v \in V} \left( \bigvee_{1 \leq j \leq k} v_j \right)
\]

3. each node has just one color:

\[
\neg(v_i \land v_j) \text{ with } v \in V, 1 \leq i < j \leq k
\]

4. neighbors have different colors:

\[
\neg(v_i \land w_i) \text{ with } (v, w) \in E, 1 \leq i \leq k
\]

2-coloring of

\[
\{a, b, c\}, \{(a, b), (b, c)\}
\]

1. \(a_1, a_2, b_1, b_2, c_1, c_2\)

2. \(a_1 \lor a_2, b_1 \lor b_2, c_1 \lor c_2\)

3. \(\neg(a_1 \land a_2), \neg(b_1 \land b_2), \neg(c_1 \land c_2)\)

4. \(\neg(a_1 \land b_1), \neg(a_2 \land b_2) \neg(b_1 \land c_1), \neg(b_2 \land c_2)\)