VL Logik (LVA-Nr. 342208), Winter Semester 2014/2015

Propositional Logic

Version 2014.5

Armin Biere (biere@jku.at)
Martina Seidl (martina.seidl@jku.at)
A proposition is a statement that is either true or false.

Example
- Alice comes to the party.
- One has to wear a shirt.
- It rains.

With connectives, propositions can be combined to complex propositions.

Example
- Alice comes to the party and Bob comes to the party, but not Cecile.
- One has to wear either a shirt or a tie.
- If it rains, the street is wet.
Propositional Logic

- language for representing, combining, and interpreting propositions
- *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: \((\neg\text{tie} \lor \text{shirt}) \land (\text{tie} \lor \text{shirt}) \land (\neg\text{tie} \lor \neg\text{shirt})\)

- atoms: tie, shirt
- connectives: ¬, ∨, ∧
- parenthesis for structuring the expression
Background

- **historical origins**: ancient Greeks
- two very basic principles:
  - **Law of Excluded Middle**: Each expression is either true or false.
  - **Law of Contradiction**: No statement is both true and false.
- very *simple* language
  - no objects, no arguments to propositions
  - no functions
  - no quantifiers
- solving is *easy* (relative to other logics)
- investigated in philosophy, mathematics, and computer science
- propositional logic in computer science:
  - description of digital circuits
  - automated verification
  - planning, scheduling, configuration problems
  - large research area in theoretical computer science
  - many applications in industry
The Language of Propositional Logic: Syntax

**Definition**

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*.

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**

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

To reduce the number of parenthesis, we use the following conventions:

- $\neg$ 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”)

In case of doubt, uses parenthesis!

**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

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

has the formula tree

![Formula Tree Diagram]
The formula

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

has the simplified formula tree

![Formula Tree Diagram]
Subformulas

Definition

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 \( \psi \).

The set of subformulas of a formula \( \phi \) is the smallest set \( S \) such that

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

Informal: A subformula is a part of a formula and is itself a formula.

Example

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/](http://fmv.jku.at/limboole/)

**input format:**

```plaintext
expr ::= iff
iff ::= implies { '<->' implies }
implies ::= or [ '<->' or | '<-' or ]
or ::= and { '|' and }
and ::= not { '&‘ not }
not ::= basic | '!‘ not
basic ::= var | '(' expr ')'
```

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

**Example**

In Limboole the formula \((a ∨ b) → (c ∧ ¬¬d)\) is represented as

\[
((a | b) \rightarrow (c & !!d))
\]
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: $(x \land y)$, $(\neg x \land x' \land \neg x'')$, $x, \neg y$
Special Formula Structures: Negation Normal Form

**Definition**

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

- Literals and truth constants are in NNF;
- $\phi \circ \psi$ ($\circ \in \{\lor, \land\}$) 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.

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 $((x \lor \neg x_1) \land (x \lor (\neg z \lor \neg x_1)))$ is in NNF but $\neg((x \lor \neg x_1) \land (x \lor (\neg z \lor \neg x_1)))$ is not in NNF.
**Special Formula Structures: Conjunctive Normal Form**

**Definition**
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 (can be set to false).

*remark:* CNF is the input of most SAT-solvers (DIMACS format).

**Example**
- $\top$
- $\bot$
- $a$
- $\neg a$
- $l_1 \land l_2 \land l_3$
- $l_1 \lor l_2 \lor l_3$
- $(a_1 \lor \neg a_2) \land (a_1 \lor b_2 \lor a_2) \land a_2$
- $((l_{11} \lor \ldots \lor l_{1m_1}) \land \ldots \land (l_{n1} \lor \ldots \lor l_{nm_n}))$
Special Formula Structures: Disjunctive Normal Form

### Definition

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 clauses.
- easy to check whether it can be satisfied (can be set to true).

### Example

- \( \top \)
- \( \bot \)
- \( a \)
- \( \neg a \)

\[
\begin{align*}
\top & \quad \llbracket l_1 \land l_2 \land l_3 \rrbracket \\
\bot & \quad \llbracket l_1 \lor l_2 \lor l_3 \rrbracket \\
a & \quad \llbracket (a_1 \land \neg a_2) \lor (a_1 \land b_2 \land a_2) \land a_2 \rrbracket \\
\neg a & \quad \llbracket ((l_{11} \land \ldots \land l_{1m_1}) \lor \ldots \lor (l_{n1} \land \ldots \land l_{nm_n})) \rrbracket
\end{align*}
\]
Conventions

In general, we use the following conventions unless stated otherwise:

- \( a, b, c, x, y, z \) denote variables.
- \( l, k \) 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 \( \phi \setminus \{C\} \).
Elements of Propositional Logic: Negation

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

<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>

truth table:

Example

- If the proposition “It rains.” is true then the negation “It does not rain.” is false.
- If proposition $a$ is true then proposition $\neg a$ is false.
- If formula $((a \lor x) \land y)$ is true then formula $\neg((a \lor x) \land y)$ is false.
- If proposition $b$ is false, then proposition $\neg b$ is true.
- If formula $((b \rightarrow y) \land z)$ is true then formula $\neg((b \rightarrow y) \land z)$ is false.
Elements of Propositional Logic: 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 $\bigwedge_{i=1}^{n} \phi_i$.
- truth table:

<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>

**Example**

- If the proposition “I want tea.” is true and if the proposition “I want cake.” is true then also “I want tea and I want cake.” is true.
- The proposition $(a \land \neg a)$ is false.
- The proposition $(\top \land a)$ is true if $a$ is true.
- The proposition $(\bot \land a)$ is 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.
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 \( \bigvee_{i=1}^{n} \phi_i \).

<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**

- The proposition \( (a \lor \neg a) \) is true.
- The proposition \( (\top \lor a) \) is true.
- The proposition \( (\bot \lor a) \) is true if \( a \) is true.
- If \( (a \to b) \) is true and \( (\neg c \to d) \) then \( (a \to b) \lor (\neg c \to d) \) is true.
- If you see "The menu includes soup or dessert." in a restaurant then this is usually not a disjunction.
Elements of Propositional Logic: Implication

- an implication is true iff the first argument is false or both arguments are true
- alternative notation for $\phi \rightarrow \psi$: $\phi \supset \psi$, $\phi IMPL \psi$
- It holds: Verum ex quodlibet. Ex falsum quodlibet.

$$
\begin{array}{c|c|c}
\phi & \psi & \phi \rightarrow \psi \\
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
$$

**truth table:**

**set view:**

Example

- If the proposition "It rains." is true and the proposition "The street is wet." is true then the statement "If it rains, the street is wet." is true.
- If the proposition "If it rains, the street is wet." is true and the statement "The street is wet." is true, it does not necessarily rain.
- The propositions $(\bot \rightarrow a)$ and $(a \rightarrow a)$ are true.
- The proposition $\top \rightarrow \phi$ is true if $\phi$ is true.
Elements of Propositional Logic: Equivalence

- binary connective
- an equivalence is true iff both elements have the same value
- alternative notation for $\phi \leftrightarrow \psi$: $\phi = \psi$, $\phi \equiv \psi$, $\phi \sim \psi$
- truth table:

<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:*

set view:

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.
Implication vs. Equivalence

In natural language, there is not always a clear distinction between equivalence and implication. The distinction comes from the context.

**equivalence:**

- Iff a student passes a course, (s)he has more than 50 points on the test.
  - To pass the course, it is necessary to have more than 50 points.
  - If a student has more than 50 points on the test then (s)he passes the test.
  - If the student does not have more than 50 points, (s)he does not pass the test.
  - If the student does not pass the test, (s)he did not get at least 50 points.

**implication:**

- If a student passes a course, (s)he has more than 50 points on the test.
  - This statement would also be true if a student fails even though having more than 50 points.
  - Having more than 50 points is necessary, but not sufficient to pass a course.
The Logic Connectives at a Glance

- The meaning of the connectives can be summarized as follows:

<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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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

- Overall, there are 16 different functions for binary connectives.
- So far, we had conjunction, disjunction, implication, equivalence.
- Further connectives:
  - $\phi \leftrightarrow \psi$ (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>$\phi \leftrightarrow \psi$</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

- **or gate**
  - A
  - B

- **xor gate**
  - A
  - B

- **nand gate**
  - A
  - B

- **nor 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

\[ c \Leftrightarrow x \land y \]

and

\[ s \Leftrightarrow x \oplus y. \]
### 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>$!\phi$</td>
<td>$not \ \phi$</td>
<td>$! \phi$</td>
</tr>
<tr>
<td>conjunction</td>
<td>$\phi \land \psi$</td>
<td>$\phi \psi \land \psi$</td>
<td>$\phi \land \psi$</td>
<td>$\phi \land \psi$</td>
<td>$\phi \land \psi$</td>
</tr>
<tr>
<td>disjunction</td>
<td>$\phi \lor \psi$</td>
<td>$\phi + \psi$</td>
<td>$\phi \lor \psi$</td>
<td>$\phi \lor \psi$</td>
<td>$\phi \lor \psi$</td>
</tr>
<tr>
<td>exclusive or</td>
<td>$\phi \oplus \psi$</td>
<td>$\phi \oplus \psi$</td>
<td>$\phi \oplus \psi$</td>
<td>$\phi \oplus \psi$</td>
<td>$-$</td>
</tr>
<tr>
<td>implication</td>
<td>$\phi \rightarrow \psi$</td>
<td>$\phi \supset \psi$</td>
<td>$\phi \rightarrow \psi$</td>
<td>$\phi \rightarrow \psi$</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>$\phi = \psi$</td>
<td>$\phi = \psi$</td>
</tr>
</tbody>
</table>

### Example

- $(a \lor (b \lor \neg c)) \leftrightarrow (\top \land ((a \rightarrow \neg b) \lor (c \lor a \lor b)))$
- $(a + (b + \bar{c})) = c ((a \supset \neg b) + (0 + a + b))$
- $(a \parallel (b \parallel \neg c)) = (c \land \parallel ((a \land \neg b) \land (false \parallel a \land b)))$
## 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 0</td>
<td></td>
<td>0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td></td>
<td>0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td></td>
<td>0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td>0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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 \( V \subseteq \mathcal{P} \cup \{\neg x | x \in \mathcal{P}\} \):
  - \( x \in V \) 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.

Example

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>((x \lor y) \land \neg 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:** \( V = \{x, \neg y, z\} \)
- **Observation:** A variable assignment determines the truth value of the formulas containing these variables.
The Language of Propositional Logic: Semantics

**Definition**

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 \([x]_\nu = \nu(x)\)
- \([\neg \phi]_\nu = 1 \iff [\phi]_\nu = 0\)
- \([\phi \lor \psi]_\nu = 1 \iff [\phi]_\nu = 1 \text{ or } [\psi]_\nu = 1\)

An assignment is called
- **satisfying** a formula \( \phi \) iff \([\phi]_\nu = 1\).
- **falsifying** a formula \( \phi \) iff \([\phi]_\nu = 0\).

An assignment satisfying a formula \( \phi \) is a **model** of \( \phi \).

An assignment falsifying a formula \( \phi \) 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 (1/2)

- formula $\phi$ is **satisfiable** iff 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
- a valid formula is called **tautology**
- 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
- an unsatisfiable formula is called **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 (2/2)

- 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 $(\bot \lor b)$
Semantic Equivalence

**Definition**

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

**Note:**
- $\iff$ is a *meta-symbol*, i.e., it is not part of the language.
- *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 $\phi \not\iff \psi$.

**Example**

- $a \lor \neg a \not\iff b \rightarrow \neg b$
- $a \lor \neg a \iff b \lor \neg b$
- $(a \lor b) \land \neg(a \lor b) \iff \bot$
- $(a \iff (b \iff c)) \iff ((a \iff b) \iff c)$
## Examples of Semantic Equivalences

<table>
<thead>
<tr>
<th>Example</th>
<th>Equivalent Example</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>
<tr>
<td>( \phi \lor \psi \Leftrightarrow \neg \phi \rightarrow \psi )</td>
<td>( \phi \rightarrow \psi \Leftrightarrow \neg \psi \rightarrow \neg \phi )</td>
<td>implications</td>
</tr>
<tr>
<td>( \phi \land \neg \phi \Leftrightarrow \bot )</td>
<td>( \phi \lor \neg \phi \Leftrightarrow \top )</td>
<td>complement</td>
</tr>
<tr>
<td>( \neg \neg \phi \Leftrightarrow \phi )</td>
<td></td>
<td>double negation</td>
</tr>
<tr>
<td>( \phi \land \top \Leftrightarrow \phi )</td>
<td>( \phi \lor \bot \Leftrightarrow \phi )</td>
<td>neutrality</td>
</tr>
<tr>
<td>( \phi \lor \top \Leftrightarrow \top )</td>
<td>( \phi \land \bot \Leftrightarrow \bot )</td>
<td></td>
</tr>
<tr>
<td>( \neg \top \Leftrightarrow \bot )</td>
<td>( \neg \bot \Leftrightarrow \top )</td>
<td></td>
</tr>
</tbody>
</table>
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 $\phi \nleftrightarrow \bot$. 
Logic Entailment

Definition

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

Informal meaning: True premises derive a true conclusion.

- $\vdash$ is a meta-symbol, i.e., it is not part of the language.
- $\phi_1, \ldots, \phi_n \vdash \psi$ 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 $\phi_1, \ldots, \phi_n \not\vdash \psi$.

Example

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

Definition

Two formulas \( \phi \) and \( \psi \) are \textit{satisfiability-equivalent} (written as \( \phi \iff_{\text{SAT}} \psi \)) 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 equivalence.
- Often sufficient for simplification rules: If the complicated formula is satisfiable then also the simplified formula is satisfiable.

Example

\textit{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.

- \( x \iff_{\text{SAT}} \top \), but \( x \not\iff \top \)
- \( (a \land b) \lor (\neg c \land a) \iff_{\text{SAT}} b \lor \neg c \), but \( (a \land b) \lor (\neg c \land a) \not\iff b \lor \neg c \)
**Representing Functions as CNFs**

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

**Solution (Representation as 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></th>
<th></th>
<th></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></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$a \lor \neg b \lor \neg c$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$a \lor \neg b \lor \neg c$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$\neg a \lor b \lor \neg c$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$\neg a \lor \neg b \lor \neg c$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$\neg a \lor \neg b \lor c$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

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

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

Solution (Representation as 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.

Example

<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>( a \land b \land \neg c )</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.

**Definition**

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.

**Example**

- $\{\neg, \wedge\}$ is functional complete.
- $\{\neg, \vee\}$ is functional complete.
- nand is functional complete.
- nor is 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**:

\(\neg (v_i \land v_j)\) with \(v \in V, 1 \leq i < j \leq k\)

4. neighbors have **different colors**:

\(\neg (v_i \land w_i)\) with \((v, w) \in E, 1 \leq i \leq k\)

**Example**

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

A lady is in one of two rooms called A and B. A tiger is also in A or B. On the door of A there is a sign: “This room contains a lady, the other room contains a tiger.” The door of room B has a sign: “The tiger and the lady are not in the same room.” One sign lies. Where is the lady, where is the tiger?

based on a puzzle by Raymond Smullyan

One possible SAT encoding:

- $\text{signOnA}$ represents that sign of room A says the truth
- $\text{signOnB}$ represents that sign of room B says the truth
- $\text{ladyInA}$ or $\text{ladyInB}$ represents that lady is in A or B respectively
- $\text{tigerInA}$ or $\text{tigerInB}$ represents that tiger is in A or B respectively
- lady is in room A or B, but not in both: $(\text{ladyInA} \lor \text{ladyInB}) \land \neg(\text{ladyInA} \land \text{ladyInB})$
- tiger is in room A or B, but not in both: $(\text{tigerInA} \lor \text{tigerInB}) \land \neg(\text{tigerInA} \land \text{tigerInB})$
- one sign lies, one sign is true: $(\text{signOnA} \leftrightarrow \neg\text{signOnB})$
- sign of room A: $\text{signOnA} \leftrightarrow (\text{ladyInA} \land \text{tigerInB})$
- sign of room B: $\text{signOnB} \leftrightarrow (\neg(\text{tigerInA} \land \text{ladyInA}) \land \neg(\text{tigerInB} \land \text{ladyInB}))$