Institute for Formal Models and Verification
Johannes Kepler University Linz

VL Logik (LVA-Nr. 342208), Winter Semester 2015/2016

Propositional Logic

Version 2015.2

Armin Biere (biere@jku.at)
Martina Seidl (martina.seidl@jku.at)

Propositions

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 ($\neg$), and ($\wedge$), or ($\vee$), ...
    - operators for construction of composite propositions
    - concise meaning
    - argument(s) and return value from Boolean domain
  - parenthesis

Example

formula of propositional logic: $(\neg \mathbf{t} \vee \mathbf{s}) \wedge (\mathbf{t} \vee \mathbf{s}) \wedge (\neg \mathbf{t} \vee \neg \mathbf{s})$

- atoms: **t**, **s**
- connectives: $\neg$, $\vee$, $\wedge$
- 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 $\circ \in \{\vee, \wedge, \leftrightarrow, \rightarrow\}$.

$\mathcal{L}$ 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:

$$\phi ::= \top \mid \bot \mid p \mid (\neg\phi) \mid (\phi \vee \phi) \mid (\phi \wedge \phi) \mid (\phi \leftrightarrow \phi) \mid (\phi \rightarrow \phi)$$

Example

- $\top$
- a
- $(\neg a)$
- $(\neg\top)$
- $(\neg(\neg a))$
- $(a_1 \vee a_2)$
- $(\neg(a \vee b))$
- $(\neg(a \leftrightarrow b))$
- $(((\neg a) \vee a') \leftrightarrow (b \rightarrow c))$
- $(((a_1 \vee a_2) \vee (a_3 \wedge \bot)) \rightarrow b)$

Rules of Precedence

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

- $\neg$ is stronger than $\wedge$
- $\wedge$ is stronger than $\vee$
- $\vee$ 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

- $\neg a \wedge b \vee c \rightarrow d \leftrightarrow f$ is the same as $(((((\neg a) \wedge b) \vee c) \rightarrow d) \leftrightarrow f)$.
- $a' \vee a'' \vee a'' \wedge b' \vee b''$ is the same as $(((a' \vee a'') \vee (a'' \wedge b')) \vee b'')$.
- $a' \wedge a'' \wedge a'' \vee b' \wedge b''$ is the same as $(((a' \wedge a'') \wedge a''') \vee (b' \wedge b''))$.

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

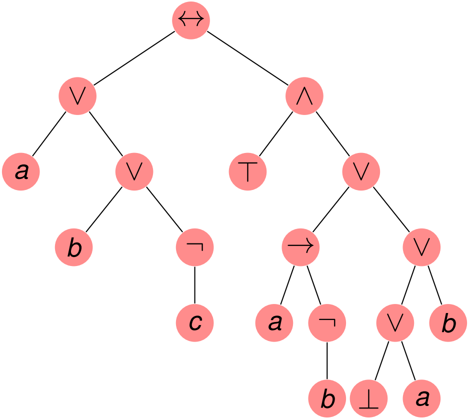
- *simplification*:
  disjunction and conjunction may be considered as *n*-ary operators, i.e., if a node *N* and its child node *C* are of the same kind of connective (conjunction / disjunction), then the children of *C* can become direct children of *N* and the *C* is removed.

Formula Tree: Example (1/2)

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

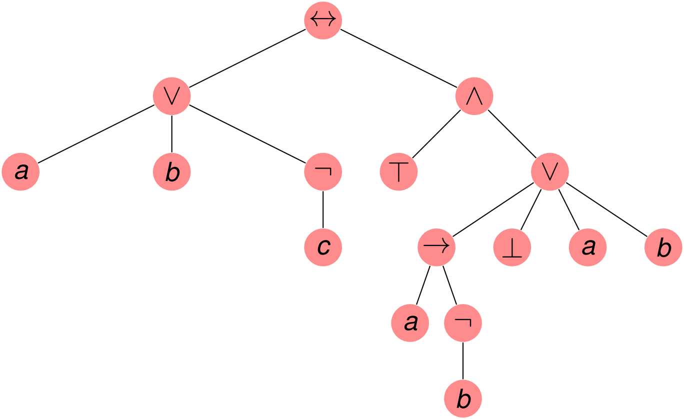


Formula Tree: Example (2/2)

The formula

$$(a \vee (b \vee \neg c)) \leftrightarrow (\top \wedge ((a \rightarrow \neg b) \vee (\bot \vee a \vee b)))$$

has the simplified formula tree



Subformulas

Definition

An *immediate subformula* is defined as follows:

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

The *set of subformulas* of a formula ϕ is the smallest set S such that

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

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

Example

The subformulas of $(a \vee b) \rightarrow (c \wedge \neg\neg d)$ are
$\{a, b, c, d, \neg d, \neg\neg d, a \vee b, c \wedge \neg\neg d, (a \vee b) \rightarrow (c \wedge \neg\neg d)\}$

Limboole

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

$$
\begin{array}{rcl}
\langle \mathit{expr} \rangle & ::= & \langle \mathit{iff} \rangle \\
\langle \mathit{iff} \rangle & ::= & \langle \mathit{implies} \rangle \mid \langle \mathit{implies} \rangle \text{ “}\texttt{<->}\text{” } \langle \mathit{implies} \rangle \\
\langle \mathit{implies} \rangle & ::= & \langle \mathit{or} \rangle \mid \langle \mathit{or} \rangle \text{ “}\texttt{->}\text{” } \langle \mathit{or} \rangle \mid \langle \mathit{or} \rangle \text{ “}\texttt{<-}\text{” } \langle \mathit{or} \rangle \\
\langle \mathit{or} \rangle & ::= & \langle \mathit{and} \rangle \mid \langle \mathit{and} \rangle \text{ “}\texttt{|}\text{” } \langle \mathit{and} \rangle \\
\langle \mathit{and} \rangle & ::= & \langle \mathit{not} \rangle \mid \langle \mathit{not} \rangle \text{ “}\texttt{\&}\text{” } \langle \mathit{not} \rangle \\
\langle \mathit{not} \rangle & ::= & \langle \mathit{basic} \rangle \mid \text{“}\texttt{!}\text{” } \langle \mathit{not} \rangle \\
\langle \mathit{basic} \rangle & ::= & \langle \mathit{var} \rangle \mid \text{“}\texttt{(}\text{” } \langle \mathit{expr} \rangle \text{ “}\texttt{)}\text{”}
\end{array}
$$

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

Example

In Limboole the formula $(a \vee b) \rightarrow (c \wedge \neg\neg d)$ is represented as

```
((a | b) -> (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: $(x \vee y)$, $(\neg x \vee x' \vee \neg x'')$, $x, \neg y$
- *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: $(x \wedge y)$, $(\neg x \wedge x' \wedge \neg x'')$, $x, \neg y$

Special Formula Structures: Negation Normal Form

Definition

Negation Normal Form (NNF) is defined as follows:

- Literals and truth constants are in NNF;
- $\phi \circ \psi$ ($\circ \in \{\vee, \wedge\}$) is in NNF iff ϕ and ψ 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 ϕ is something else than a variable or a constant.
- it does not contain NAND, NOR, XOR, equivalence, and implication connectives.

Example

The formula

$((x \vee \neg x_1) \wedge (x \vee (\neg z \vee \neg x_1)))$

is in NNF but

$\neg((x \vee \neg x_1) \wedge (x \vee (\neg z \vee \neg x_1)))$

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
- $\neg a$
- $l_1 \wedge l_2 \wedge l_3$
- $l_1 \vee l_2 \vee l_3$
- $(a_1 \vee \neg a_2) \wedge (a_1 \vee b_2 \vee a_2) \wedge a_2$
- $((l_{11} \vee \ldots \vee l_{1m_1}) \wedge \ldots \wedge (l_{n1} \vee \ldots \vee l_{nm_n}))$

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
- $\neg a$
- $l_1 \wedge l_2 \wedge l_3$
- $l_1 \vee l_2 \vee l_3$
- $(a_1 \wedge \neg a_2) \vee (a_1 \wedge b_2 \wedge a_2) \vee a_2$
- $((l_{11} \wedge \ldots \wedge l_{1m_1}) \vee \ldots \vee (l_{n1} \wedge \ldots \wedge l_{nm_n}))$

Conventions

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

- a, b, c, x, y, z denote *variables*.
- l, k denote *literals*.
- ϕ, ψ, γ denote *arbitrary formulas*.
- C, D denote *clauses or cubes* (clear from context).
- *Clauses* are also written as sets.
  - $(l_1 \vee \ldots \vee l_n) = \{l_1, \ldots l_n\}$.
  - To add a literal l to clause C, we write $C \cup \{l\}$.
  - To remove a literal l from clause C, we write $C \backslash \{l\}$.
- *Formulas in CNF* are also written as sets of sets.
  - $((l_{11} \vee \ldots \vee l_{1m_1}) \wedge \ldots \wedge (l_{n1} \vee \ldots \vee l_{nm_n})) = \{\{l_{11}, \ldots l_{1m_1}\}, \ldots, \{l_{n1}, \ldots l_{nm_n}\}\}$.
  - To add a clause C to CNF ϕ, we write $\phi \cup \{C\}$.
  - To remove a clause C from CNF ϕ, we write $\phi \backslash \{C\}$.

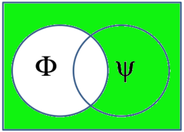
Elements of Propositional Logic: Negation

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

truth table:

ϕ	$\neg\phi$
0	**1**
1	**0**

set view:



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 $((a \vee x) \wedge y)$ is true then formula $\neg((a \vee x) \wedge y)$ is false.
- If proposition b is false, then proposition $\neg b$ is true.
- If formula $((b \rightarrow y) \wedge z)$ is true then formula $\neg((b \rightarrow y) \wedge z)$ is false.

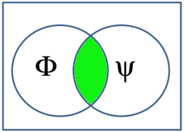
Elements of Propositional Logic: Conjunction

- a conjunction is true iff both arguments are true
- alternative notation for $\phi \wedge \psi$: $\phi \& \psi$, $\phi\psi, \phi * \psi, \phi \cdot \psi, \phi \mathit{AND} \psi$
- For $(\phi_1 \wedge \ldots \wedge \phi_n)$ we also write $\bigwedge_{i=1}^{n} \phi_i$.
- truth table:

truth table:

ϕ	ψ	$\phi \wedge \psi$
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

set view:



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 $(a \wedge \neg a)$ is false.
- The proposition $(\top \wedge a)$ is true if a is true.
- The proposition $(\bot \wedge a)$ is false.
- If $(a \vee b)$ is true and $(\neg c \vee d)$ is true then $(a \vee b) \wedge (\neg c \vee d)$ is true.

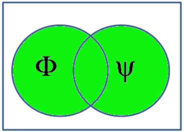
Elements of Propositional Logic: Disjunction

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

truth table:

ϕ	ψ	$\phi \vee \psi$
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

set view:



Example

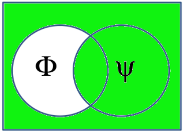
- The proposition $(a \vee \neg a)$ is true.
- The proposition $(\top \vee a)$ is true.
- The proposition $(\bot \vee a)$ is true if a is true.
- If $(a \to b)$ is true and $(\neg c \to d)$ then $(a \to b) \vee (\neg c \to d)$ is true.
- If you see "The menu includes soup or dessert." in a restaurant 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 $\phi \to \psi$: $\phi \supset \psi$, $\phi IMPL \psi$
- It holds: Verum ex quodlibet. Ex falsum quodlibet.

truth table:

ϕ	ψ	$\phi \to \psi$
0	0	1
0	1	1
1	0	0
1	1	1

set view: 

Example

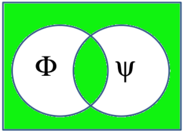
- If the proposition "It rains." is true and the proposition "The street is wet." is true then the statement "If it rains, the street is wet." is true.
- If the proposition "If it rains, the street is wet." is true and the statement "The street is wet." is true, it does not necessarily rain.
- The propositions $(\perp \to a)$ and $(a \to a)$ are true.
- The proposition $\top \to \phi$ is true if ϕ 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:

truth table:

ϕ	ψ	$\phi \leftrightarrow \psi$
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

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:

ϕ	ψ	$\top$	$\bot$	$\neg\phi$	$\phi \land \psi$	$\phi \lor \psi$	$\phi \to \psi$	$\phi \leftrightarrow \psi$	$\phi \oplus \psi$	$\phi \uparrow \psi$	$\phi \downarrow \psi$
0	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**
0	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
1	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**
1	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**

Example

ϕ	ψ	$\neg(\neg\phi \land \neg\psi)$	$\neg\phi \lor \psi$	$(\phi \to \psi) \land (\psi \to \phi)$
0	**0**	**0**	**1**	**1**
0	**1**	**1**	**1**	**0**
1	**0**	**1**	**0**	**0**
1	**1**	**1**	**1**	**1**

Observation: connectives can be expressed by other connectives.

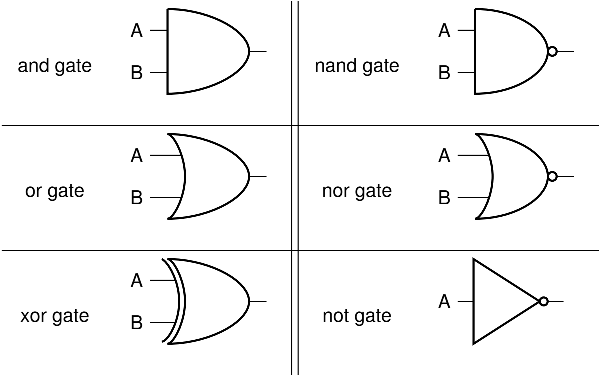
Other Connectives

- Overall, there are 16 different functions for binary connectives.
- So far, we had conjunction, disjunction, implication, equivalence.
- Further connectives:
  - $\phi \not\leftrightarrow \psi$ (also $\oplus$, *xor*, antivalence)
  - $\phi \uparrow \psi$ (*nand*, Sheffer Stroke Function)
  - $\phi \downarrow \psi$ (*nor*, Pierce Function)

ϕ	ψ	$\phi \not\leftrightarrow \psi$	$\phi \uparrow \psi$	$\phi \downarrow \psi$
0	**0**	**0**	**1**	**1**
0	**1**	**1**	**1**	**0**
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**0**	**0**

- 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



Example of a Digital Circuit: Half Adder

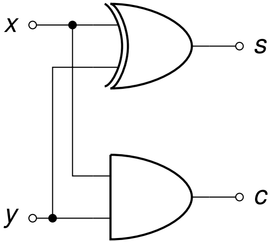
x	y	c	s
0	**0**	**0**	**0**
0	**1**	**0**	**1**
1	**0**	**0**	**1**
1	**1**	**1**	**0**

From the truth table, we see that

$$c \Leftrightarrow x \wedge y$$

and

$$s \Leftrightarrow x \oplus y.$$



Different Notations

operator	logic	circuits	Verilog C/C++/Java/C#	VHDL	Limboole
1	$\top$	1	*true*	1	–
0	$\bot$	0	*false*	0	–
negation	$\neg\phi$	$\bar{\phi}$ $\quad -\phi$	$!\phi$	*not* ϕ	$!\,\phi$
conjunction	$\phi \wedge \psi$	$\phi\psi \quad \phi \cdot \psi$	ϕ && ψ	ϕ *and* ψ	ϕ & ψ
disjunction	$\phi \vee \psi$	$\phi + \psi$	ϕ || ψ	ϕ *or* ψ	ϕ | ψ
exclusive or	$\phi \not\leftrightarrow \psi$	$\phi \oplus \psi$	ϕ != ψ	ϕ *xor* ψ	–
implication	$\phi \rightarrow \psi$	$\phi \supset \psi$	–	–	ϕ -> ψ
equivalence	$\phi \leftrightarrow \psi$	$\phi = \psi$	ϕ == ψ	ϕ *xnor* ψ	ϕ <-> ψ

Example

- $(a \vee (b \vee \neg c)) \leftrightarrow (\top \wedge ((a \rightarrow \neg b) \vee (c \vee a \vee b)))$
- $(a + (b + \bar{c})) = c\ ((a \supset -b) + (0 + a + b))$
- (*a* || (*b* || !*c*)) == (*c* && ((! *a* || ! *b*) || (*false* || *a* || *b*)))

All 16 Binary Functions

ϕ	ψ	constant 0	nor					xor	nand	and	equivalence		implication			or	constant 1
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Assignment

- A variable can be assigned one of two values from the two-valued domain $\mathbb{B}$, where $\mathbb{B} = \{\mathbf{1}, \mathbf{0}\}$.
- The mapping $\nu : \mathcal{P} \to \mathbb{B}$ is called *assignment*, where $\mathcal{P}$ is the set of atomic propositions.
- We sometimes write an assignment ν as set $V \subseteq \mathcal{P} \cup \{\neg x | x \in \mathcal{P}\}$:
  - $x \in V$ iff $\nu(x) = \mathbf{1}$
  - $\neg x \in V$ iff $\nu(x) = \mathbf{0}$
- For n variables, there are 2^n assignments possible.
- An assignment corresponds to one line in the truth table.

Example

x	y	z	$(x \vee y) \wedge \neg z$
0	**0**	**0**	**0**
0	**0**	**1**	**0**
0	**1**	**0**	**1**
0	**1**	**1**	**0**
1	**0**	**0**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**1**
1	**1**	**1**	**0**

- One assignment: $\nu(x) = \mathbf{1}, \nu(y) = \mathbf{0}, \nu(z) = \mathbf{1}$
- 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 $\nu : \mathcal{P} \to \mathbb{B}$, the interpretation $[.]_\nu : \mathcal{L} \to \mathbb{B}$ is defined by:

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

- An assignment is called
  - *satisfying* a formula ϕ iff $[\phi]_\nu = \mathbf{1}$.
  - *falsifying* a formula ϕ iff $[\phi]_\nu = \mathbf{0}$.
- An assignment satisfying a formula ϕ is a *model* of ϕ.
- An assignment falsifying a formula ϕ is a *counter-model* of ϕ.

Example

For formula $((x \vee y) \wedge \neg z)$,

- $\{x, y, z\}$ is a counter-model,
- $\{x, y, \neg z\}$ is a model.

Properties of Propositional Formulas (1/2)

- formula ϕ is *satisfiable* iff exists interpretation $[.]_\nu$ with $[\phi]_\nu = \mathbf{1}$
  check with `limboole -s`
- formula ϕ is *valid* iff for all interpretations $[.]_\nu$ it holds that $[\phi]_\nu = \mathbf{1}$
  check with `limboole`
- a valid formula is called *tautology*
- formula ϕ is *refutable* iff exists interpretation $[.]_\nu$ with $[\phi]_\nu = \mathbf{0}$
  check with `limboole`
- formula ϕ is *unsatisfiable* iff $[\phi]_\nu = \mathbf{0}$ for all interpretations $[.]_\nu$
  check with `limboole -s`
- an unsatisfiable formula is called *contradiction*

Example

- $\top$ is valid.
- $\bot$ is unsatisfiable.
- $(a \vee \neg b) \wedge (\neg a \vee b)$ is refutable.
- $a \to b$ is satisfiable.
- $a \leftrightarrow \neg a$ is a contradiction.
- $(a \vee \neg b) \wedge (\neg a \vee b)$ 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: $(a \vee b) \wedge (\neg a \vee c)$
- valid, not refutable $\top \vee (a \wedge \neg a)$
- not valid, refutable $(\bot \vee b)$

Semantic Equivalence

Definition

Two formula ϕ and ψ are *semantic equivalent* (written as $\phi \Leftrightarrow \psi$) iff forall interpretations $[.]_\nu$ it holds that $[\phi]_\nu = [\psi]_\nu$.

Note:

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

Example

- $a \vee \neg a \not\Leftrightarrow b \rightarrow \neg b$
- $a \vee \neg a \Leftrightarrow b \vee \neg b$
- $(a \vee b) \wedge \neg(a \vee b) \Leftrightarrow \perp$
- $(a \leftrightarrow (b \leftrightarrow c)) \Leftrightarrow ((a \leftrightarrow b) \leftrightarrow c)$

Examples of Semantic Equivalences

$\phi \land \psi \Leftrightarrow \psi \land \phi$	$\phi \lor \psi \Leftrightarrow \psi \lor \phi$	commutativity
$\phi \land (\psi \land \gamma) \Leftrightarrow (\phi \land \psi) \land \gamma$	$\phi \lor (\psi \lor \gamma) \Leftrightarrow (\phi \lor \psi) \lor \gamma$	associativity
$\phi \land (\phi \lor \psi) \Leftrightarrow \phi$	$\phi \lor (\phi \land \psi) \Leftrightarrow \phi$	absorption
$\phi \land (\psi \lor \gamma) \Leftrightarrow (\phi \land \psi) \lor (\phi \land \gamma)$	$\phi \lor (\psi \land \gamma) \Leftrightarrow (\phi \lor \psi) \land (\phi \lor \gamma)$	distributivity
$\neg(\phi \land \psi) \Leftrightarrow \neg\phi \lor \neg\psi$	$\neg(\phi \lor \psi) \Leftrightarrow \neg\phi \land \neg\psi$	laws of De Morgan
$\phi \leftrightarrow \psi \Leftrightarrow (\phi \rightarrow \psi) \land (\psi \rightarrow \phi)$	$\phi \leftrightarrow \psi \Leftrightarrow (\phi \land \psi) \lor (\neg\phi \land \neg\psi)$	synt. equivalence
$\phi \lor \psi \Leftrightarrow \neg\phi \rightarrow \psi$	$\phi \rightarrow \psi \Leftrightarrow \neg\psi \rightarrow \neg\phi$	implications
$\phi \land \neg\phi \Leftrightarrow \bot$	$\phi \lor \neg\phi \Leftrightarrow \top$	complement
$\neg\neg\phi \Leftrightarrow \phi$		double negation
$\phi \land \top \Leftrightarrow \phi$	$\phi \lor \bot \Leftrightarrow \phi$	neutrality
$\phi \lor \top \Leftrightarrow \top$	$\phi \land \bot \Leftrightarrow \bot$	
$\neg\top \Leftrightarrow \bot$	$\neg\bot \Leftrightarrow \top$	

Further Connections between Formulas

- A formula ϕ is valid iff $\neg\phi$ is unsatisfiable.
- A formula ϕ is satisfiable iff $\neg\phi$ is not valid.
- The formulas ϕ and ψ are equivalent iff $\phi \leftrightarrow \psi$ is valid.
- The formulas ϕ and ψ are equivalent iff $\neg(\phi \leftrightarrow \psi)$ is unsatisfiable.
- A formula ϕ is satisfiable iff $\phi \not\leftrightarrow \bot$.

Logic Entailment

Definition

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

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 $(\phi_1 \wedge \ldots \wedge \phi_n) \rightarrow \psi$ is valid, i.e., we can express semantics by means of syntactics.
- If $\phi_1, \ldots \phi_n$ do not entail ψ, we write $\phi_1, \ldots \phi_n \not\models \psi$.

Example

- $a \models a \vee b$
- $a, b \models a \wedge b$
- $a, a \rightarrow b \models b$
- $\models a \vee \neg a$
- $\not\models a \wedge \neg a$
- $\bot \models a \wedge \neg a$

Satisfiability Equivalence

Definition

Two formulas ϕ and ψ are *satisfiability-equivalent* (written as $\phi \Leftrightarrow_{SAT} \psi$) 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

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.

- $x \Leftrightarrow_{SAT} \top$, but $x \not\Leftrightarrow \top$
- $(a \land b) \lor (\neg c \land a) \Leftrightarrow_{SAT} b \lor \neg c$, but $(a \land b) \lor (\neg c \land a) \not\Leftrightarrow b \lor \neg c$

Representing Functions as CNFs

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

Solution (Representation as CNF):

1. Represent each assignment ν where ϕ has value **0** as clause:
   - If variable x is **1** in ν, add $\neg x$ to clause.
   - If variable x is **0** in ν, add x to clause.
2. Connect all clauses by conjunction.

Example

a	b	c	ϕ	clauses
0	**0**	**0**	**0**	$a \vee b \vee c$
0	**0**	**1**	**1**	
0	**1**	**0**	**1**	
0	**1**	**1**	**0**	$a \vee \neg b \vee \neg c$
1	**0**	**0**	**1**	
1	**0**	**1**	**0**	$\neg a \vee b \vee \neg c$
1	**1**	**0**	**0**	$\neg a \vee \neg b \vee c$
1	**1**	**1**	**1**	

$$\phi = (a \vee b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$$

Representing Functions as DNFs

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

Solution (Representation as DNF):

1. Represent each assignment ν where ϕ has value **1** as cube:
   - If variable x is **1** in ν, add x to cube.
   - If variable x is **0** in ν, add $\neg x$ to cube.
2. Connect all cubes by disjunction.

Example

a	b	c	ϕ	cubes
0	**0**	**0**	**0**	
0	**0**	**1**	**1**	$\neg a \wedge \neg b \wedge c$
0	**1**	**0**	**1**	$\neg a \wedge b \wedge \neg c$
0	**1**	**1**	**0**	
1	**0**	**0**	**1**	$a \wedge \neg b \wedge \neg c$
1	**0**	**1**	**0**	
1	**1**	**0**	**0**	
1	**1**	**1**	**1**	$a \wedge b \wedge c$

$$\phi = (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge c)$$

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^{2^n} functions of arity n.
- A function of arity n has 2^n different combinations of arguments (lines in the truth table).
- A functions 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 ... node $v \in V$ has color j $(1 \leq j \leq k)$
2. each node has *a color*:
$$\bigwedge_{v \in V} \Big(\bigvee_{1 \leq j \leq k} v_j \Big)$$
3. each node has *just one color*:
$\neg(v_i \wedge v_j)$ with $v \in V, 1 \leq i < j \leq k$
4. neighbors have *different colors*:
$\neg(v_i \wedge w_i)$ 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. $a_1 \vee a_2, b_1 \vee b_2, c_1 \vee c_2$
3. $\neg(a_1 \wedge a_2), \neg(b_1 \wedge b_2), \neg(c_1 \wedge c_2)$
4. $\neg(a_1 \wedge b_1), \neg(a_2 \wedge b_2)$
$\neg(b_1 \wedge c_1), \neg(b_2 \wedge c_2)$

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:

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