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

JOHANNES KEPLER UNIVERSITY LINZ

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 ($\neg$), 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
- 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 $\circ \in \{\vee, \wedge, \leftrightarrow, \rightarrow\}$.

$\mathcal{L}$ 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:

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

Example:

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

Rules of Precedence

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

- $\neg$ 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")

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.

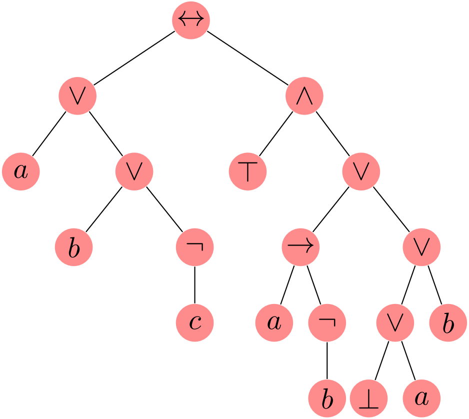
JKU

Formula Tree: Example (1/2)

The formula

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

has the formula tree

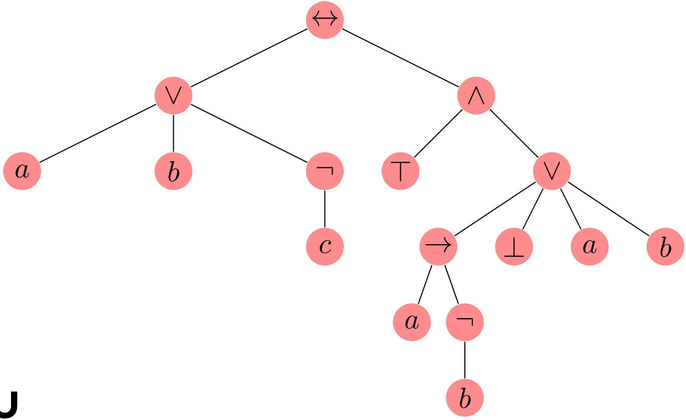


Formula Tree: Example (2/2)

The formula

$$(a \lor (b \lor \neg c)) \leftrightarrow (\top \land ((a \to \neg b) \lor (\bot \lor a \lor b)))$$

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 ϕ.
- formula $\phi \circ \psi$ ($\circ \in \{\wedge, \vee, \leftrightarrow, \rightarrow\}$) has immediate subformulas ϕ and ψ.

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

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

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

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:

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

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

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 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): $\perp$
  - 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: $(x \wedge y)$, $(\neg x \wedge x' \wedge \neg x''), x, \neg y$

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 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.

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 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.

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
- $\perp$ if it contains no cubes
- easy to check whether it can be satisfied

Examples for CNF and DNF

Examples CNF

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

Examples DNF

- $\top$
- $\bot$
- 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

we use the following conventions unless stated otherwise:

- a, b, c, x, y, z denote *variables* and 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 $\phi \backslash \{C\}$

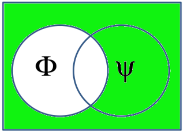
Negation

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

truth table:

ϕ	$\neg\phi$
0	**1**
1	**0**

set view:



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

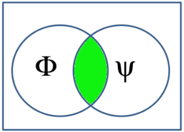
Conjunction

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

truth table:

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

set view:



Example:

- $(a \wedge \neg a)$ is always false.
- $(\top \wedge a)$ is true if a is true. $(\bot \wedge \phi)$ is always 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.

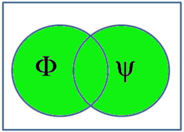
Disjunction

- a disjunction is true iff at least one of the arguments is true
- alternative notation for $\phi \vee \psi$: $\phi | \psi, \phi + \psi, \phi OR \psi$
- 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:

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

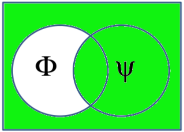
Implication

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

truth table:

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

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.
- $(\perp \rightarrow a)$ and $(a \rightarrow a)$ are always true. $\top \rightarrow \phi$ is true if ϕ is true.

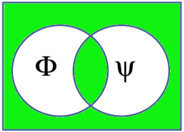
Equivalence

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

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.

The Logic Connectives at a Glance

ϕ	ψ	$\top$	$\bot$	$\neg\phi$	$\phi \wedge \psi$	$\phi \vee \psi$	$\phi \rightarrow \psi$	$\phi \leftrightarrow \psi$	$\phi \oplus \psi$	$\phi \uparrow \psi$	$\phi \downarrow \psi$
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	$\neg(\neg\phi \wedge \neg\psi)$	$\neg\phi \vee \psi$	$(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$
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.

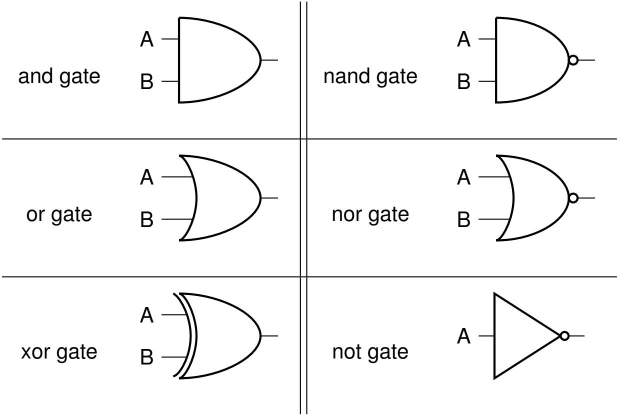
Other Connectives

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

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**

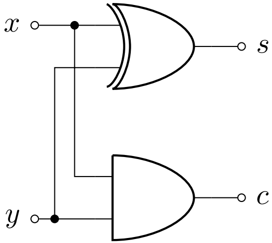
- nor and nand can express every other boolean function (i.e., they are functional complete)
- often used for building digital circuits (like processors)

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	$\neg\phi$	$\bar{\phi}$ $-\phi$	$!\phi$	$not\ \phi$	`!` ϕ
conjunction	$\phi \wedge \psi$	$\phi\psi$ $\phi \cdot \psi$	$\phi\ \&\&\ \psi$	$\phi\ and\ \psi$	ϕ `&` ψ
disjunction	$\phi \vee \psi$	$\phi + \psi$	$\phi\ \vert\vert\ \psi$	$\phi\ or\ \psi$	ϕ `|` ψ
exclusive or	$\phi \not\leftrightarrow \psi$	$\phi \oplus \psi$	ϕ != ψ	$\phi\ xor\ \psi$	–
implication	$\phi \rightarrow \psi$	$\phi \supset \psi$	–	–	ϕ `->` ψ
equivalence	$\phi \leftrightarrow \psi$	$\phi = \psi$	$\phi\ ==\ \psi$	$\phi\ xnor\ \psi$	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 with $V \subseteq \mathcal{P} \cup \{\neg x | x \in \mathcal{P}\}$ such that
  - $x \in V$ 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

JKU

Assignment: 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.

Semantics of Propositional Logic

Given assignment $\nu : \mathcal{P} \to \mathbb{B}$, the interpretation $[.]_\nu : \mathcal{L} \to \mathbb{B}$ 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**
- $[\phi \vee \psi]_\nu =$ **1** iff $[\phi]_\nu =$ **1** or $[\psi]_\nu =$ **1**

Satisfying/Falsifying Assigments

- An assignment is called
  - *satisfying* a formula ϕ iff $[\phi]_\nu = \mathbf{1}$.
  - *falsifying* a formula ϕ iff $[\phi]_\nu = \mathbf{0}$.
- A satisfying assignment for ϕ is a *model* of ϕ.
- A falsifying assignment for 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 (1/3)

- formula ϕ is *satisfiable* iff
  there 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`
- 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`

Properties of Propositional Formulas (2/3)

- a valid formula is called *tautology*
- an unsatisfiable formula is called *contradiction*

Example:

- $\top$ is valid.
- $\bot$ is unsatisfiable.
- $(a \lor \neg b) \land (\neg a \lor b)$ is refutable.
- $a \to b$ 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: $(a \vee b) \wedge (\neg a \vee c)$
- valid, not refutable $\top \vee (a \wedge \neg a)$; not valid, refutable $(\bot \vee b)$

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 $\phi \not\leftrightarrow \bot$.

Simple Algorithm for Satisfiability Checking

Algorithm: evaluate

Data: formula ϕ
Result: **1** iff ϕ is satisfiable

if *ϕ contains a variable x* **then**
  pick $v \in \{\top, \bot\}$
  /* replace x by truth constant v, evaluate resulting formula */
  if *evaluate($\phi[x|v]$)* **then return** ***1***;
  else return *evaluate($\phi[x|\overline{v}]$)* ;
else
  switch ϕ **do**
    case $\top$ **do return** ***1***;
    case $\bot$ **do return** ***0***;
    case $\neg\psi$ **do return** ***!*** *evaluate(ψ)* /* true iff ψ is false */ ;
    case $\psi' \wedge \psi''$ **do**
      return *evaluate(ψ') && evaluate(ψ'')* /* true iff both ψ' and ψ'' are true */
    case $\psi' \vee \psi''$ **do**
      return *evaluate(ψ') || evaluate(ψ'')* /* true iff ψ' or ψ'' is true */

Semantic Equivalence

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

- $\Leftrightarrow$ 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:

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

Examples of Semantic Equivalences (1/2)

$\phi \wedge \psi \Leftrightarrow \psi \wedge \phi$	$\phi \vee \psi \Leftrightarrow \psi \vee \phi$	commutativity
$\phi \wedge (\psi \wedge \gamma) \Leftrightarrow (\phi \wedge \psi) \wedge \gamma$	$\phi \vee (\psi \vee \gamma) \Leftrightarrow (\phi \vee \psi) \vee \gamma$	associativity
$\phi \wedge (\phi \vee \psi) \Leftrightarrow \phi$	$\phi \vee (\phi \wedge \psi) \Leftrightarrow \phi$	absorption
$\phi \wedge (\psi \vee \gamma) \Leftrightarrow (\phi \wedge \psi) \vee (\phi \wedge \gamma)$	$\phi \vee (\psi \wedge \gamma) \Leftrightarrow (\phi \vee \psi) \wedge (\phi \vee \gamma)$	distributivity
$\neg(\phi \wedge \psi) \Leftrightarrow \neg\phi \vee \neg\psi$	$\neg(\phi \vee \psi) \Leftrightarrow \neg\phi \wedge \neg\psi$	laws of De Morgan
$\phi \leftrightarrow \psi \Leftrightarrow (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$	$\phi \leftrightarrow \psi \Leftrightarrow (\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)$	synt. equivalence

Examples of Semantic Equivalences (2/2)

$\phi \lor \psi \Leftrightarrow \neg\phi \to \psi$	$\phi \to \psi \Leftrightarrow \neg\psi \to \neg\phi$	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
$\phi \lor \top \Leftrightarrow \top$	$\phi \land \bot \Leftrightarrow \bot$	
$\neg\top \Leftrightarrow \bot$	$\neg\bot \Leftrightarrow \top$	

Logic Entailment

Let $\phi_1, \dots \phi_n, \psi$ be propositional formulas. Then $\phi_1, \dots \phi_n$ *entail* ψ (written as $\phi_1, \dots, \phi_n \models \psi$) iff $[\phi_1]_\nu = \mathbf{1}, \dots [\phi_n]_\nu = \mathbf{1}$ 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.
- $\phi_1, \dots \phi_n \models \psi$ iff $(\phi_1 \wedge \dots \wedge \phi_n) \rightarrow \psi$ is valid, i.e., we can express semantics by means of syntactics.
- If $\phi_1, \dots \phi_n$ do not entail ψ, we write $\phi_1, \dots \phi_n \not\models \psi$.

Example:

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

Satisfiability Equivalence

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

Representing Functions as CNFs

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

Solution (in 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.

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 (in 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.

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**	$a \wedge b \wedge c$

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

JKU

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**.

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.

$\{\neg, \wedge\}$, $\{\neg, \vee\}$, {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 ... node $v \in V$ has color j ($1 \leq j \leq k$)
2. each node has *a color*:
$$\bigwedge_{v \in V} (\bigvee_{1 \leq j \leq k} v_j)$$
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 $(v, w) \in E, 1 \leq i \leq k$

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. $\neg(a_1 \wedge b_1), \neg(a_2 \wedge b_2)\ \neg(b_1 \wedge c_1), \neg(b_2 \wedge c_2)$

JKU