PROPOSITIONAL LOGIC II

VL Logik: WS 2017/18

Martina Seidl (martina.seidl@jku.at),
Armin Biere (biere@jku.at)
Institut für Formale Modelle und Verifikation

JOHANNES KEPLER
UNIVERSITY LINZ

Example: Party Planning

We want to plan a party.

Unfortunately, the selection of the guests is not straight forward.

We have to consider the following rules.

1. If two people are married, we have to invite them both or none of them. Alice is married to Bob and Cecile is married to David.
2. If we invite Alice then we also have to invite Cecile.
   Cecile does not care if we invite Alice but not her.
3. David and Eva can't stand each other, so it is not possible to invite both.
4. We want to invite Bob and Fred.

Question: Can we find a guest list?

Party Planning with Propositional Logic

- propositional variables:
  inviteAlice, inviteBob, inviteCecile, inviteDavid, inviteEva, inviteFred
- constraints:
  1. invite married: $\text{inviteAlice} \leftrightarrow \text{inviteBob}$, $\text{inviteCecile} \leftrightarrow \text{inviteDavid}$
  2. if Alice then Cecile: $\text{inviteAlice} \rightarrow \text{inviteCecile}$
  3. either David or Eva: $\neg(\text{inviteEva} \leftrightarrow \text{inviteDavid})$
  4. invite Bob and Fred: $\text{inviteBob} \wedge \text{inviteFred}$
- encoding in propositional logic:

$$(\text{inviteAlice} \leftrightarrow \text{inviteBob}) \wedge (\text{inviteCecile} \leftrightarrow \text{inviteDavid}) \wedge (\text{inviteAlice} \rightarrow \text{inviteCecile}) \wedge \neg(\text{inviteEva} \leftrightarrow \text{inviteDavid}) \wedge \text{inviteBob} \wedge \text{inviteFred}$$

JKU

A Puzzle

A lady is in one of the 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

A Puzzle

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

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 <u>propositional formulas</u>.

Excursus: Backus-Naur Form (BNF)

- notation technique for describing the syntax of a language
- elements:
  - non-terminal symbols (variables): enclosed in brackets $\langle\rangle$
  - ::= indicates the definition of a non-terminal symbol
  - the symbol | means “or”
  - all other symbols stand for themselves (sometimes they are quoted, e.g., “->”)

example: definition of the language of decimal numbers in BNF:

$$\begin{aligned}\langle number\rangle &::= \langle integer\rangle \text{ “.” } \langle integer\rangle\\ \langle integer\rangle &::= \langle digit\rangle \mid \langle digit\rangle\ \langle integer\rangle\\ \langle digit\rangle &::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid 0\end{aligned}$$

some words: 0.0, 1.1, 123.546, 01.10000, . . .

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

Example:

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

Formula Tree

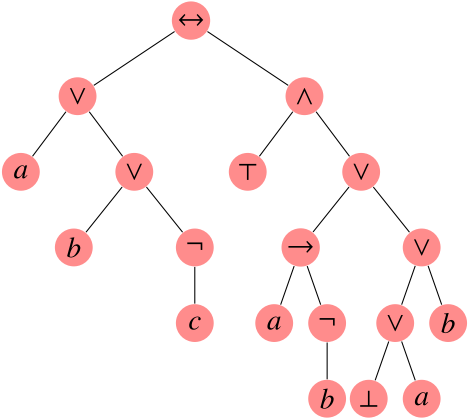
- formulas have a tree structure
  - inner nodes: connectives
  - leaves: truth constants, variables
- default: inner nodes have one child node (negation) or two nodes as children (other connectives).
- tree structure reflects the use of parenthesis
- simplification:
  disjunction and conjunction may be considered as n-ary operators,
  i.e., if a node N and its child node C are of the same kind of connective (conjunction / disjunction), then the children of C can become direct children of N and the C is removed.

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

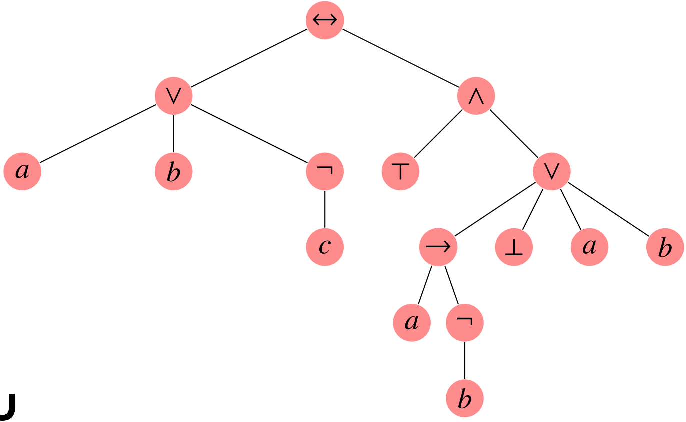


Formula Tree: Example (2/2)

The formula

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

has the simplified formula tree



Subformulas

An immediate subformula is defined as follows:

- truth constants and atoms have no immediate subformula.
- only immediate subformula of $\neg\phi$ is 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:

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

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

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 $\mathrm{var}(l) = x$.
  - 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: $(x \wedge y), (\neg x \wedge x' \wedge \neg x''), x, \neg y$

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 <u>conjunctive normal form</u> (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

<u>remark</u>: CNF is the input of most SAT-solvers (DIMACS format)

Disjunctive Normal Form (DNF)

A propositional formula is in <u>disjunctive normal form (DNF)</u> 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 $\phi \backslash \{C\}$

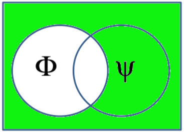
Negation

- 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.

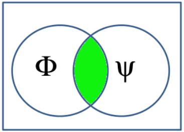
Conjunction

- 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 $\bigwedge_{i=1}^{n} \phi_i$.

<u>truth table</u>:

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

<u>set view</u>:



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.

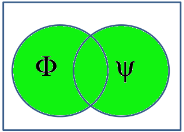
Disjunction

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

truth table:

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

set view:



Example:

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

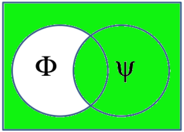
Implication

- 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.

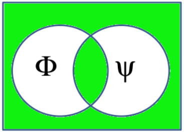
Equivalence

- 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	$\top$	$\bot$	$\neg\phi$	$\phi \land \psi$	$\phi \lor \psi$	$\phi \to \psi$	$\phi \leftrightarrow \psi$	$\phi \oplus \psi$	$\phi \uparrow \psi$	$\phi \downarrow \psi$
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.

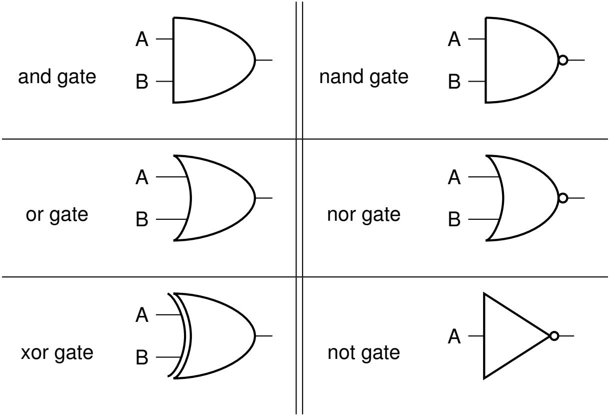
Other Connectives

- 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**

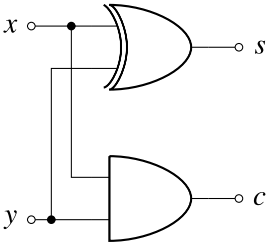
- nor and nand can express every other boolean function (i.e., they are functional complete)
- often used for building digital circuits (like processors)

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	$\neg\phi$	$\bar{\phi}$ $\quad -\phi$	$!\phi$	$not\ \phi$	$!\phi$
conjunction	$\phi \wedge \psi$	$\phi\psi \quad \phi \cdot \psi$	$\phi\ \&\&\ \psi$	$\phi\ and\ \psi$	$\phi\ \&\ \psi$
disjunction	$\phi \vee \psi$	$\phi + \psi$	$\phi \mid\mid \psi$	$\phi\ or\ \psi$	$\phi \mid \psi$
exclusive or	$\phi \not\leftrightarrow \psi$	$\phi \oplus \psi$	$\phi\ {!}{=}\ \psi$	$\phi\ xor\ \psi$	–
implication	$\phi \rightarrow \psi$	$\phi \supset \psi$	–	–	ϕ -> ψ
equivalence	$\phi \leftrightarrow \psi$	$\phi = \psi$	$\phi\ ==\ \psi$	$\phi\ xnor\ \psi$	ϕ <-> ψ

Example:

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

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 <u>assignment</u>, 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

Semantics of Propositional Logic

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

- $[\top]_\nu = \mathbf{1}$, $[\bot]_\nu = \mathbf{0}$
- 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 $[\psi]_\nu = \mathbf{1}$

What about the other connectives?

JKU

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

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

JKU

Properties of Propositional Formulas (1/3)

- formula ϕ is <u>satisfiable</u> iff
  there exists interpretation $[.]_\nu$ with $[\phi]_\nu = \mathbf{1}$
  check with `limboole -s`
- formula ϕ is <u>valid</u> iff
  for all interpretations $[.]_\nu$ it holds that $[\phi]_\nu = \mathbf{1}$
  check with `limboole`
- formula ϕ is <u>refutable</u> iff
  exists interpretation $[.]_\nu$ with $[\phi]_\nu = \mathbf{0}$
  check with `limboole`
- formula ϕ is <u>unsatisfiable</u> iff
  $[\phi]_\nu = \mathbf{0}$ for all interpretations $[.]_\nu$
  check with `limboole -s`

Properties of Propositional Formulas (2/3)

- a valid formula is called <u>tautology</u>
- an unsatisfiable formula is called <u>contradiction</u>

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 (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 $\psi' \land \psi''$ **do**
      return evaluate(ψ') && evaluate(ψ'')   /* true iff both ψ' and ψ'' are true */
    case $\psi' \lor \psi''$ **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:

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

JKU

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)

$\phi \vee \psi \Leftrightarrow \neg\phi \rightarrow \psi$	$\phi \rightarrow \psi \Leftrightarrow \neg\psi \rightarrow \neg\phi$	implications
$\phi \wedge \neg\phi \Leftrightarrow \bot$	$\phi \vee \neg\phi \Leftrightarrow \top$	complement
$\neg\neg\phi \Leftrightarrow \phi$		double negation
$\phi \wedge \top \Leftrightarrow \phi$	$\phi \vee \bot \Leftrightarrow \phi$	neutrality
$\phi \vee \top \Leftrightarrow \top$	$\phi \wedge \bot \Leftrightarrow \bot$	
$\neg\top \Leftrightarrow \bot$	$\neg\bot \Leftrightarrow \top$	

Logic Entailment

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:

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

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**	$\neg a \land \neg b \land c$
0	**1**	**0**	**1**	$\neg a \land b \land \neg c$
0	**1**	**1**	**0**	
1	**0**	**0**	**1**	$a \land \neg b \land \neg c$
1	**0**	**1**	**0**	
1	**1**	**0**	**0**	
1	**1**	**1**	**1**	$a \land b \land c$

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

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^{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.

JKU