FORMULAS IN CONJUNCTIVE NORMAL FORM (CNF)

VL Logic, Lecture 1, WS 20/21

Armin Biere, Martina Seidl

Institute for Formal Models and Verification
Johannes Kepler University Linz

JOHANNES KEPLER
UNIVERSITY LINZ

A VERY, VERY SHORT PRIMER ON (PROPOSITIONAL) LOGIC

VL Logic

Part I: Propositional Logic

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

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

Defining a Language: Syntax

What do expressions (words, sentences) of a language look like?

- **set of words**: basic building blocks
- **grammar**: rules for composing sentences from words and smaller sentences
- sometimes multiple (equivalent) representations
  - different goals (user-friendliness vs processability)

Defining a Language: Syntax

What do expressions (words, sentences) of a language look like?

- **set of words**: basic building blocks
- **grammar**: rules for composing sentences from words and smaller sentences
- sometimes multiple (equivalent) representations
  - different goals (user-friendliness vs processability)

example:

- syntactically incorrect formula: $\wedge(a \vee \wedge b) \wedge (\neg a \vee \neg b)$
- syntactically correct propositional formula (over variables a and b):
  $(a \vee b) \wedge (\neg a \vee \neg b)$
- different representation:

```
 1  2 0
-1 -2 0
```

Defining a Language: Semantics

What do expressions mean?

- evaluation of syntactically correct expressions
- logic-based languages have a concise semantics
  - no ambiguities
  - no vagueness

Defining a Language: Semantics

What do expressions mean?

- evaluation of syntactically correct expressions
- logic-based languages have a concise semantics
  - no ambiguities
  - no vagueness

Let propositional variable a be true and propositional variable b be false.
Then the truth value of the propositional formula

$$(a \vee b) \wedge (\neg a \vee \neg b)$$

is true.

Using a Language: Pragmatics

What does an expression stand for?

- many application problems can be formulated as logical reasoning problems
- interpretation of expression dependends on context and user
  1. encoding of an application problem to be solved
  2. evaluation of expression gives solution for application problem

Using a Language: Pragmatics

What does an expression stand for?

- many application problems can be formulated as logical reasoning problems
- interpretation of expression dependends on context and user
    1. encoding of an application problem to be solved
    2. evaluation of expression gives solution for application problem

In the formula

$$(a \lor b) \land (\neg a \lor \neg b)$$

a could stand “invite David” and b could stand for “invite Eva”. The formula is true if only one of the two persons is invited.

Automated Reasoning and Inferences

- Logical languages allow the inference of new knowledge (“reasoning”).
- For reasoning, a logic provides various sets of **rules** (calculi).
- Reasoning is often based on certain syntactical patterns.

example: (modus ponens)

$$\frac{\text{x holds.} \quad \text{If x holds, then also y holds.}}{\text{y holds.}}$$

Some Remarks on Inferences

A system is inconsistent if we can infer that a statement holds and that a statement does not hold at the same time.

Assume we have modelled the following system

- A comes to the party.
- B comes to the party.
- If A comes to the party, then B does not come to the party.

With the **modus ponens**, we can infer that B does not come to the party.
So, we have some inconsistency in our party model.

Logics in this Lecture

In this lecture, we consider different logic-based languages:

- **part 1: propositional logic (SAT)**
  - simple language: only atoms and connectives
  - low expressiveness, low complexity

Logics in this Lecture

In this lecture, we consider different logic-based languages:

- **part 1: propositional logic (SAT)**
  - simple language: only atoms and connectives
  - low expressiveness, low complexity
- **part 2: first-order logic (predicate logic)**
  - rich language: predicates, functions, terms, quantifiers
  - great power of expressiveness, high complexity

Logics in this Lecture

In this lecture, we consider different logic-based languages:

- **part 1: propositional logic (SAT)**
  - simple language: only atoms and connectives
  - low expressiveness, low complexity
- **part 2: first-order logic (predicate logic)**
  - rich language: predicates, functions, terms, quantifiers
  - great power of expressiveness, high complexity
- **part 3: satisfiability modulo theories (SMT)**
  - customizable language: user decides
  - as much expressiveness as required
    as much complexity as necessary

Logics in this Lecture

In this lecture, we consider different logic-based languages:

- **part 1: propositional logic (SAT)**
  - simple language: only atoms and connectives
  - low expressiveness, low complexity
- **part 2: first-order logic (predicate logic)**
  - rich language: predicates, functions, terms, quantifiers
  - great power of expressiveness, high complexity
- **part 3: satisfiability modulo theories (SMT)**
  - customizable language: user decides
  - as much expressiveness as required
    as much complexity as necessary

SYNTAX OF PROPOSITIONAL FORMULAS IN CNF

VL Logic

Part I: Propositional Logic

JOHANNES KEPLER
UNIVERSITY LINZ

Truth Constants, Variables, Literals

- **truth constant**:
  - *true* (also verum, top): $\top$
  - *false* (also falsum, bottom): $\bot$

Truth Constants, Variables, Literals

- **truth constant**:
  - *true* (also verum, top): $\top$
  - *false* (also falsum, bottom): $\bot$
- **propositional variable** (also atom, atomic proposition):
  - proposition without any further internal structure
  - we denote variables by symbols x, y, z, a, b, c, p, q (possibly with subscript)

Truth Constants, Variables, Literals

- **truth constant**:
  - *true* (also verum, top): $\top$
  - *false* (also falsum, bottom): $\bot$
- **propositional variable** (also atom, atomic proposition):
  - proposition without any further internal structure
  - we denote variables by symbols x, y, z, a, b, c, p, q (possibly with subscript)
- **literal**:
  - (negated) truth constant: $\top, \bot, \neg\top, \neg\bot$
  - (negated) propositional variable $x, \neg x, y, \neg y, \ldots$
  - we denote literals by symbols l, k (possibly with subscript)

Clauses

A **clause** is a disjunction ($\vee$) of literals.

Clauses

A **clause** is a disjunction ($\vee$) of literals.

- **examples**: $x \vee y$ $\quad$ $x \vee y \vee \neg z$ $\quad$ $\neg\bot \vee x \vee a \vee \top \vee \neg x$

Clauses

A **clause** is a disjunction ($\vee$) of literals.

- **examples**: $x \vee y$ $\quad$ $x \vee y \vee \neg z$ $\quad$ $\neg\bot \vee x \vee a \vee \top \vee \neg x$
- **size of a clause**: number of its literals

Clauses

A **clause** is a disjunction ($\vee$) of literals.

- **examples**: $x \vee y$ $\quad$ $x \vee y \vee \neg z$ $\quad$ $\neg\bot \vee x \vee a \vee \top \vee \neg x$
- **size of a clause**: number of its literals
- **special clauses**

empty clause	(size 0):	also written as $\bot$ or $\emptyset$
unary clause	(size 1):	$\neg\bot$ $\quad$ x $\quad$ $\neg z$
binary clauses	(size 2):	$x \vee y$
ternary clauses	(size 3):	$x \vee y \vee \neg z$

Clauses

A **clause** is a disjunction ($\vee$) of literals.

- **examples**: $x \vee y$ $\quad$ $x \vee y \vee \neg z$ $\quad$ $\neg\bot \vee x \vee a \vee \top \vee \neg x$
- **size of a clause**: number of its literals
- **special clauses**

empty clause	(size 0):	also written as $\bot$ or $\emptyset$
unary clause	(size 1):	$\neg\bot$ $\quad$ x $\quad$ $\neg z$
binary clauses	(size 2):	$x \vee y$
ternary clauses	(size 3):	$x \vee y \vee \neg z$

- for $(l_1 \vee \ldots \vee l_n)$ we also write $\bigvee_{i=1}^{n} l_i$

Clauses

A **clause** is a disjunction ($\vee$) of literals.

- **examples**: $x \vee y$ $\quad$ $x \vee y \vee \neg z$ $\quad$ $\neg\bot \vee x \vee a \vee \top \vee \neg x$
- **size of a clause**: number of its literals
- **special clauses**

empty clause	(size 0):	also written as $\bot$ or $\emptyset$
unary clause	(size 1):	$\neg\bot$ $\quad$ x $\quad$ $\neg z$
binary clauses	(size 2):	$x \vee y$
ternary clauses	(size 3):	$x \vee y \vee \neg z$

- for $(l_1 \vee \ldots \vee l_n)$ we also write $\bigvee_{i=1}^{n} l_i$
- we denote clauses by symbols C, D (possibly with subscript)

Propositional Formulas in CNF

A **propositional formula in conjunctive normal form** is a conjunction ($\wedge$) of clauses.

Propositional Formulas in CNF

A **propositional formula in conjunctive normal form** is a conjunction ($\wedge$) of clauses.

- **examples**:
  - $(x \vee \top) \wedge (y \vee \neg z) \wedge (\neg y \vee \neg x)$
  - $(\neg x \vee y \vee \neg z) \wedge z$
  - $(x \vee \neg y) \wedge (x \vee \neg y \vee z) \wedge (y \vee \neg z)$
  - $((l_{11} \vee \ldots \vee l_{1m_1}) \wedge \ldots \wedge (l_{n1} \vee \ldots \vee l_{nm_n}))$

Propositional Formulas in CNF

A **propositional formula in conjunctive normal form** is a conjunction ($\wedge$) of clauses.

- **examples**:
  - $(x \vee \top) \wedge (y \vee \neg z) \wedge (\neg y \vee \neg x)$
  - $(\neg x \vee y \vee \neg z) \wedge z$
  - $(x \vee \neg y) \wedge (x \vee \neg y \vee z) \wedge (y \vee \neg z)$
  - $((l_{11} \vee \ldots \vee l_{1m_1}) \wedge \ldots \wedge (l_{n1} \vee \ldots \vee l_{nm_n}))$
- for $(C_1 \wedge \ldots \wedge C_n)$ we also write $\bigwedge_{i=1}^{n} C_i$.

Propositional Formulas in CNF

A **propositional formula in conjunctive normal form** is a conjunction ($\wedge$) of clauses.

- **examples**:
  - $(x \vee \top) \wedge (y \vee \neg z) \wedge (\neg y \vee \neg x)$
  - $(\neg x \vee y \vee \neg z) \wedge z$
  - $(x \vee \neg y) \wedge (x \vee \neg y \vee z) \wedge (y \vee \neg z)$
  - $((l_{11} \vee \ldots \vee l_{1m_1}) \wedge \ldots \wedge (l_{n1} \vee \ldots \vee l_{nm_n}))$
- for $(C_1 \wedge \ldots \wedge C_n)$ we also write $\bigwedge_{i=1}^{n} C_i$.
- we denote formulas by symbols ϕ, ψ (possibly with subscript)

Alternative Notations

in the literature, different symbols might be used for building propositional formulas

	in this lecture	alternative notations
truth constant “true”	$\top$	1 t true
truth constant “false”	$\bot$	0 f false
negation	$\neg x$	$!x$ $\overline{x}$ $-x$ NOTx
disjunction	$k \vee l$	$l \mid\mid k$ $l + k$ l OR k
conjunction	$C \wedge D$	$C \&\& D$ $C * D$ C AND D

Summary

We introduced a sublanguage of the language of propositional logic:
propositional formulas in conjunctive normal form (CNF)

- formula in CNF: conjunction of clauses
- clause: disjunction of literals
- literal: (negated) variable / (negated) truth constant

Any propositional formula can be translated into CNF.

SEMANTICS OF PROPOSITIONAL FORMULAS IN CNF

VL Logic

Part I: Propositional Logic

JOHANNES KEPLER UNIVERSITY LINZ

Truth Constants and Variables

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

Truth Constants and Variables

- 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 $\nu : \mathcal{P} \rightarrow \mathbb{B}$ is called **assignment**, where $\mathcal{P}$ is the set of variables of a formula

Truth Constants and Variables

- 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 $v : \mathcal{P} \to \mathbb{B}$ is called **assignment**, where $\mathcal{P}$ is the set of variables of a formula
- we sometimes write an assignment v as set V with $V \subseteq \mathcal{P} \cup \{\neg x \mid x \in \mathcal{P}\}$ such that
  - $x \in V$ iff $v(x) = \mathbf{1}$
  - $\neg x \in V$ iff $v(x) = \mathbf{0}$

Truth Constants and Variables

- 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 $v : \mathcal{P} \rightarrow \mathbb{B}$ is called **assignment**, where $\mathcal{P}$ is the set of variables of a formula
- we sometimes write an assignment v as set V with $V \subseteq \mathcal{P} \cup \{\neg x \mid x \in \mathcal{P}\}$ such that
  - $x \in V$ iff $v(x) = \mathbf{1}$
  - $\neg x \in V$ iff $v(x) = \mathbf{0}$
- for n variables, there are 2^n assignments possible

Truth Constants and Variables

- 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 variables of a formula
- we sometimes write an assignment ν as set V with $V \subseteq \mathcal{P} \cup \{\neg x \mid x \in \mathcal{P}\}$ 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
- $\top$ is always true and $\perp$ is always false

Negation Operator

- unary connective $\neg$ (operator with exactly one operand)
- semantics: flipping the truth value of its operand (under a given assignment)

truth table:

x	$\neg x$
0	**1**
1	**0**

examples:

- $\neg\bot$ is true.
- $\neg\top$ is false.
- Given assignment ν and variable a with $\nu(a) = \mathbf{1}$ then $\neg a$ is false under ν.
- Given assignment ν and variable a with $\nu(a) = \mathbf{0}$ then $\neg a$ is true under ν.

Binary Disjunction Operator

- binary operator $\vee$ (operator with exactly two operands)
- semantics: true iff at least one operand is true

truth table:

l	k	$l \vee k$
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

examples:

- $(a \vee \neg a)$ is always true.
- $(\top \vee a)$ is always true.
- The truth value of $(\perp \vee a)$ depends on the truth value of a.

Properties of Disjunction

- **commutative**: $k \lor l$ is true iff $l \lor k$ is true, i.e.,

  $k \lor l$ and $l \lor k$ are *equivalent*: $k \lor l \Leftrightarrow l \lor k$

Properties of Disjunction

- **commutative**: $k \vee l$ is true iff $l \vee k$ is true, i.e.,

  $k \vee l$ and $l \vee k$ are *equivalent*: $k \vee l \Leftrightarrow l \vee k$
- **idempotent**: $l \vee l$ is true iff l is true, i.e.,

  $l \vee l$ and l are *equivalent*: $l \vee l \Leftrightarrow l$

Properties of Disjunction

- **commutative**: $k \vee l$ is true iff $l \vee k$ is true, i.e.,

  $k \vee l$ and $l \vee k$ are *equivalent*: $k \vee l \Leftrightarrow l \vee k$
- **idempotent**: $l \vee l$ is true iff l is true, i.e.,

  $l \vee l$ and l are *equivalent*: $l \vee l \Leftrightarrow l$
- **associative**: $l_1 \vee (l_2 \vee l_3)$ is true iff $(l_1 \vee l_2) \vee l_3$ is true, i.e.,

  $l_1 \vee (l_2 \vee l_3)$ and $(l_1 \vee l_2) \vee l_3$ are *equivalent*: $l_1 \vee (l_2 \vee l_3) \Leftrightarrow (l_1 \vee l_2) \vee l_3$

Properties of Disjunction

- **commutative**: $k \vee l$ is true iff $l \vee k$ is true, i.e.,

  $k \vee l$ and $l \vee k$ are *equivalent*: $k \vee l \Leftrightarrow l \vee k$
- **idempotent**: $l \vee l$ is true iff l is true, i.e.,

  $l \vee l$ and l are *equivalent*: $l \vee l \Leftrightarrow l$
- **associative**: $l_1 \vee (l_2 \vee l_3)$ is true iff $(l_1 \vee l_2) \vee l_3$ is true, i.e.,

  $l_1 \vee (l_2 \vee l_3)$ and $(l_1 \vee l_2) \vee l_3$ are *equivalent*: $l_1 \vee (l_2 \vee l_3) \Leftrightarrow (l_1 \vee l_2) \vee l_3$

clauses are also written as sets

- $(l_1 \vee l_2 \vee \ldots \vee l_{n-1} \vee l_n) = \{l_1, l_2, \ldots, l_{n-1}, l_n\}$
- to add a literal l to clause C, we write $C \cup \{l\}$
- to remove a literal l from clause C, we write $C \backslash \{l\}$

Clause

- a clause is true iff at least one of the literals is true

truth table:

l_1	$\dots$	l_n	$l_1 \vee l_2 \vee \dots \vee l_n$
0	$\dots$	**0**	**0**
0	$\dots$	**1**	**1**
	$\dots$		**1**
1	$\dots$	**0**	**1**
1	$\dots$	**1**	**1**

Clause

- a clause is true iff at least one of the literals is true

truth table:

l_1	$\ldots$	l_n	$l_1 \vee l_2 \vee \ldots \vee l_n$
0	$\ldots$	**0**	**0**
0	$\ldots$	**1**	**1**
	$\ldots$		**1**
1	$\ldots$	**0**	**1**
1	$\ldots$	**1**	**1**

- the empty clause is always false

Binary Conjunction Operator

- binary operator $\wedge$ (operator with exactly two operands)
- semantics: a conjunction is true iff both operands are true

truth table:

C	D	$C \wedge D$
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

examples:

- $(a \wedge \neg a)$ 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.

Properties of Conjunction

- commutative:

$$C \land D \Leftrightarrow D \land C$$

- idempotent:

$$C \land C \Leftrightarrow C$$

- associative:

$$C_1 \land (C_2 \land C_3) \Leftrightarrow (C_1 \land C_2) \land C_3$$

Properties of Conjunction

- commutative:
$$C \wedge D \Leftrightarrow D \wedge C$$
- idempotent:
$$C \wedge C \Leftrightarrow C$$
- associative:
$$C_1 \wedge (C_2 \wedge C_3) \Leftrightarrow (C_1 \wedge C_2) \wedge C_3$$

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 $\phi \backslash \{C\}$

CNF Formulas

- a formula in CNF is true iff all of its clauses are true

truth table:

C_1	$\ldots$	C_n	$C_1 \wedge C_2 \wedge \ldots \wedge C_n$
0	...	0	0
0	...	1	0
	...		0
1	...	0	0
1	...	1	1

CNF Formulas

- a formula in CNF is true iff all of its clauses are true

truth table:

C_1	$\ldots$	C_n	$C_1 \wedge C_2 \wedge \ldots \wedge C_n$
0	...	**0**	**0**
0	...	**1**	**0**
	...		**0**
1	...	**0**	**0**
1	...	**1**	**1**

- the empty CNF formula is always true

Truth Table: Example

the truth table of CNF formula $(x \vee y) \wedge \neg z$:

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

Truth Table: Example

the truth table of CNF formula $(x \lor y) \land \neg z$:

	x	y	z	$x \lor y$	$\neg z$	$(x \lor y) \land \neg z$
	0	0	0	0	1	0
	0	0	1	0	0	0
	0	1	0	1	1	1
	0	1	1	1	0	0
$\rightarrow$	1	0	0	1	1	1
	1	0	1	1	0	0
	1	1	0	1	1	1
	1	1	1	1	0	0

- one assignment: $v(x) = \mathbf{1}, v(y) = \mathbf{0}, v(z) = \mathbf{0}$
- alternative notation: $V = \{x, \neg y, \neg z\}$

Rules of Precedence

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

Rules of Precedence

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

example:

- $\neg a \vee b \wedge \neg c \vee d$
  - is the same as $(\neg a) \vee (b \wedge (\neg c)) \vee d$,
  - but not as $((\neg a) \vee b) \wedge ((\neg c) \vee d)$

Rules of Precedence

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

example:

- $\neg a \vee b \wedge \neg c \vee d$
  - is the same as $(\neg a) \vee (b \wedge (\neg c)) \vee d$,
  - but not as $((\neg a) \vee b) \wedge ((\neg c) \vee d)$

$\Rightarrow$ put clauses into parentheses!

Semantics of Propositional Logic

Let $\mathcal{P}$ be a set of atoms and $\mathcal{L}$ be the set of all CNFs over $\mathcal{P}$.

Given assignment $\nu : \mathcal{P} \to \mathbb{B}$, we define the **interpretation function** $[.]_\nu : \mathcal{L} \to \mathbb{B}$ by:

- $[\top]_\nu = \mathbf{1}$, $[\bot]_\nu = \mathbf{0}$
- $[x]_\nu = \nu(x)$ (where $x \in \mathcal{P}$, i.e., x is a variable)
- $[\neg p]_\nu = \mathbf{1}$ iff $[p]_\nu = \mathbf{0}$ (where $p \in \mathcal{P} \cup \{\top, \bot\}$)
- $[C]_\nu = \mathbf{1}$ (where C is a clause) iff
  there is at least one literal l with $l \in C$ and $[l]_\nu = \mathbf{1}$
- $[\phi]_\nu = \mathbf{1}$ (where ϕ is in CNF) iff
  for all clauses $C \in \phi$ it holds that $[C]_\nu = \mathbf{1}$

SAT: THE SATISFIABLITY PROBLEM

VL Logic

Part I: Propositional Logic

JOHANNES KEPLER
UNIVERSITY LINZ

Satisfying/Falsifying Assignments

- an assignment v is called
  - **satisfying** a formula ϕ iff $[\phi]_v = \mathbf{1}$
  - **falsifying** a formula ϕ iff $[\phi]_v = \mathbf{0}$
- 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 (1/2)

- formula ϕ is **satisfiable** iff
  there exists an assignment ν with $[\phi]_\nu = \mathbf{1}$
- formula ϕ is **valid** iff
  for all assignments ν it holds that $[\phi]_\nu = \mathbf{1}$
- formula ϕ is **refutable** iff
  there exists an assignment ν with $[\phi]_\nu = \mathbf{0}$
- formula ϕ is **unsatisfiable** iff
  for all assignments ν it holds that $[\phi]_\nu = \mathbf{0}$

Properties of Propositional Formulas (2/2)

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

example:

- $\top$ is valid.
- $a \vee \neg a$ is a tautology.
- $(a \vee \neg b) \wedge (\neg a \vee b)$ is refutable.
- $\bot$ is unsatisfiable.
- $a \wedge \neg a$ is a contradiction.
- $(a \vee \neg b) \wedge (\neg a \vee b)$ is satisfiable.

SAT: The Boolean Satisfiability Problem

Given a propositional formula ϕ.
Is there an assignment that satisfies ϕ?

SAT: The Boolean Satisfiability Problem

Given a propositional formula ϕ.
Is there an assignment that satisfies ϕ?

formulation for formulas in CNF: can we find an assignment such that each clause contains at least one true literal?

Satisfiability Checking

- oldest **NP**-complete problem
  - checking a solution (does the assignment satisfies the formula?) is easy (polynomial effort)
  - finding a solution is difficult (probably exponential in the worst case)
- many practical applications (used in industry)
- efficient SAT solvers (solving tools) are available

THE SAT SOLVER LIMBOOLE

VL Logic

Part I: Propositional Logic

JOHANNES KEPLER
UNIVERSITY LINZ

SAT-Solver Limboole

- available at `http://fmv.jku.at/limboole`
- input:[1]
  - variables are strings over letters, digits and – _ . [ ] $ @
  - negation symbol $\neg$ is `!`
  - disjunction symbol $\vee$ is `|`
  - conjunction symbol $\wedge$ is &

example

$(a \vee b \vee \neg c) \wedge (\neg a \vee b) \wedge c$ is represented as

`(a | b | !c) & (!a | b) & c`

[1] For now, we will only use subset of the language supported by Limboole.

Tool Demo