

Propositional Logic

Basics

Syntax of propositional logic

Definition

An **atomic formula** (or **atom**) has the form A_i where $i = 1, 2, 3, \dots$

Formulas are defined inductively:

- ▶ \perp (“False”) and \top (“True”) are formulas
- ▶ All atomic formulas are formulas
- ▶ For all formulas F , $\neg F$ is a formula.
- ▶ For all formulas F and G , $(F \circ G)$ is a formula, where $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$

\neg	is called	negation
\wedge	is called	conjunction
\vee	is called	disjunction
\rightarrow	is called	implication
\leftrightarrow	is called	bi-implication

Parentheses

Precedence of logical operators in decreasing order:

$$\neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow$$

Operators with higher precedence bind more strongly.

Example

Instead of $(A \rightarrow ((B \wedge \neg(C \vee D)) \vee E))$

we can write $A \rightarrow B \wedge \neg(C \vee D) \vee E$.

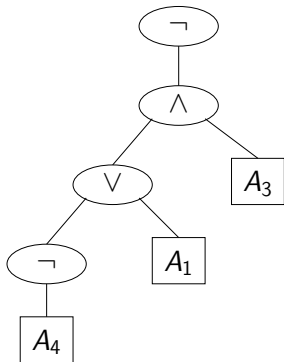
Outermost parentheses can be dropped.

Syntax tree of a formula

Every formula can be represented by a syntax tree.

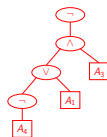
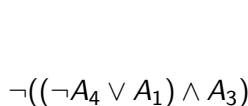
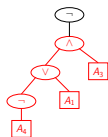
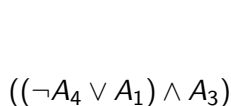
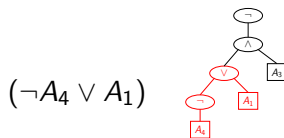
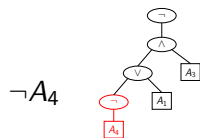
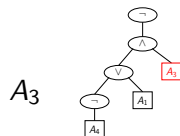
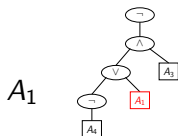
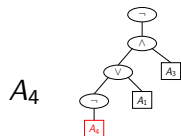
Example

$$F = \neg((\neg A_4 \vee A_1) \wedge A_3)$$



Subformulas

The **subformulas** of a formula are the formulas corresponding to the subtrees of its syntax tree.



Induction on formulas

Proof by induction on the structure of a formula:

In order to prove some property $\mathcal{P}(F)$ for all formulas F it suffices to prove the following:

- ▶ Base cases:
prove $\mathcal{P}(\perp)$, prove $\mathcal{P}(\top)$, and prove $\mathcal{P}(A_i)$ for all atoms A_i
- ▶ Induction step for \neg :
prove $\mathcal{P}(\neg F)$ under the induction hypothesis $\mathcal{P}(F)$
- ▶ Induction step for all $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$:
prove $\mathcal{P}(F \circ G)$ under the induction hypotheses $\mathcal{P}(F)$ and $\mathcal{P}(G)$

Operators that are merely abbreviations need not be considered!

Semantics of propositional logic (I)

The elements of the set $\{0, 1\}$ are called **truth values**.
(You may call 0 “false” and 1 “true”)

An **assignment** is a function $\mathcal{A} : Atoms \rightarrow \{0, 1\}$
where *Atoms* is the set of all atoms.

We extend \mathcal{A} to a function $\hat{\mathcal{A}} : Formulas \rightarrow \{0, 1\}$

Semantics of propositional logic (II)

$$\hat{\mathcal{A}}(A_i) = \mathcal{A}(A_i)$$

$$\hat{\mathcal{A}}(\neg F) = \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{\mathcal{A}}(F \wedge G) = \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 1 \text{ and } \hat{\mathcal{A}}(G) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{\mathcal{A}}(F \vee G) = \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 1 \text{ or } \hat{\mathcal{A}}(G) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{\mathcal{A}}(F \rightarrow G) = \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 0 \text{ or } \hat{\mathcal{A}}(G) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Instead of $\hat{\mathcal{A}}$ we simply write \mathcal{A}

Using arithmetic: $\mathcal{A}(F \wedge G) = \min(\mathcal{A}(F), \mathcal{A}(G))$

$\mathcal{A}(F \vee G) = \max(\mathcal{A}(F), \mathcal{A}(G))$

Abbreviations

$A, B, C,$
 $P, Q, R,$ or ... instead of $A_1, A_2, A_3 \dots$

$$F_1 \leftrightarrow F_2 \text{ abbreviates } (F_1 \wedge F_2) \vee (\neg F_1 \wedge \neg F_2)$$
$$\bigvee_{i=1}^n F_i \text{ abbreviates } (\dots ((F_1 \vee F_2) \vee F_3) \vee \dots \vee F_n)$$
$$\bigwedge_{i=1}^n F_i \text{ abbreviates } (\dots ((F_1 \wedge F_2) \wedge F_3) \wedge \dots \wedge F_n)$$

Special cases:

$$\bigvee_{i=1}^0 F_i = \bigvee \emptyset = \perp \qquad \bigwedge_{i=1}^0 F_i = \bigwedge \emptyset = \top$$

Truth tables (I)

We can compute \hat{A} with the help of **truth tables**.

\neg	A	A	\vee	B	A	\wedge	B
1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1
		1	1	0	1	0	0
		1	1	1	1	1	1

A	\rightarrow	B	A	\leftrightarrow	B
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	1	1	1	1	1

Coincidence Lemma

Lemma

Let \mathcal{A}_1 and \mathcal{A}_2 be two assignments.

If $\mathcal{A}_1(A_i) = \mathcal{A}_2(A_i)$ for all atoms A_i in some formula F ,
then $\mathcal{A}_1(F) = \mathcal{A}_2(F)$.

Proof.

Exercise. □

Models

If $\mathcal{A}(F) = 1$ then we write $\mathcal{A} \models F$
and say F is true under \mathcal{A}
or \mathcal{A} is a model of F

If $\mathcal{A}(F) = 0$ then we write $\mathcal{A} \not\models F$
and say F is false under \mathcal{A}
or \mathcal{A} is not a model of F

Validity and satisfiability

Definition (Validity)

A formula F is **valid** (or a **tautology**) if every assignment is a model of F .

We write $\models F$ if F is valid, and $\not\models F$ otherwise.

Definition (Satisfiability)

A formula F is **satisfiable** if it has at least one model; otherwise F is **unsatisfiable**.

A (finite or infinite!) set of formulas S is **satisfiable** if there is an assignment that is a model of every formula in S .

Exercise

	Valid	Satisfiable	Unsatisfiable
A			

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$			

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$			

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x		

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x
$A \rightarrow \neg A$			

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x
$A \rightarrow \neg A$		x	

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x
$A \rightarrow \neg A$		x	
$A \rightarrow (B \rightarrow A)$			

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x
$A \rightarrow \neg A$		x	
$A \rightarrow (B \rightarrow A)$	x		

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x
$A \rightarrow \neg A$		x	
$A \rightarrow (B \rightarrow A)$	x	x	

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x
$A \rightarrow \neg A$		x	
$A \rightarrow (B \rightarrow A)$	x	x	
$A \rightarrow (A \rightarrow B)$			

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x
$A \rightarrow \neg A$		x	
$A \rightarrow (B \rightarrow A)$	x	x	
$A \rightarrow (A \rightarrow B)$		x	

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x
$A \rightarrow \neg A$		x	
$A \rightarrow (B \rightarrow A)$	x	x	
$A \rightarrow (A \rightarrow B)$		x	
$A \leftrightarrow \neg A$			

Exercise

	Valid	Satisfiable	Unsatisfiable
A		x	
$A \vee B$		x	
$A \vee \neg A$	x	x	
$A \wedge \neg A$			x
$A \rightarrow \neg A$		x	
$A \rightarrow (B \rightarrow A)$	x	x	
$A \rightarrow (A \rightarrow B)$		x	
$A \leftrightarrow \neg A$			x

Exercise

Which of the following statements are true?

	Y	C.ex.
If F is valid then F is satisfiable		

Exercise

Which of the following statements are true?

	Y	C.ex.
If F is valid then F is satisfiable	Y	

Exercise

Which of the following statements are true?

	Y	C.ex.
If F is valid then F is satisfiable	Y	
If F is satisfiable then $\neg F$ is satisfiable		

Exercise

Which of the following statements are true?

	Y	C.ex.
If F is valid then F is satisfiable	Y	
If F is satisfiable then $\neg F$ is satisfiable		T

Exercise

Which of the following statements are true?

	Y	C.ex.
If F is valid then F is satisfiable	Y	
If F is satisfiable then $\neg F$ is satisfiable		T
If F is valid then $\neg F$ is unsatisfiable		

Exercise

Which of the following statements are true?

	Y	C.ex.
If F is valid then F is satisfiable	Y	
If F is satisfiable then $\neg F$ is satisfiable		T
If F is valid then $\neg F$ is unsatisfiable	Y	

Exercise

Which of the following statements are true?

	Y	C.ex.
If F is valid then F is satisfiable	Y	
If F is satisfiable then $\neg F$ is satisfiable		T
If F is valid then $\neg F$ is unsatisfiable	Y	
If F is unsatisfiable then $\neg F$ is unsatisfiable		

Exercise

Which of the following statements are true?

	Y	C.ex.
If F is valid then F is satisfiable	Y	
If F is satisfiable then $\neg F$ is satisfiable		T
If F is valid then $\neg F$ is unsatisfiable	Y	
If F is unsatisfiable then $\neg F$ is unsatisfiable		\perp

Mirroring principle

all propositional formulas

valid formulas G	satisfiable but not valid formulas F $\neg F$	unsatisfiable formulas $\neg G$
------------------------------	--	---

Consequence (aka entailment)

Definition

A formula G is a (semantic) consequence of a set of formulas M if every model \mathcal{A} of all $F \in M$ is also a model of G .

We also say that M entails G and write $M \models G$.

In a nutshell:

“Every model of M is a model of G .”

Example

$A \vee B, A \rightarrow B, B \wedge R \rightarrow \neg A, R \models (R \wedge \neg A) \wedge B$

Consequence

Example

$$\underbrace{A \vee B, A \rightarrow B, B \wedge R \rightarrow \neg A, R}_M \models (R \wedge \neg A) \wedge B$$

Proof:

Assume $\mathcal{A} \models F$ for all $F \in M$.

We need to prove $\mathcal{A} \models (R \wedge \neg A) \wedge B$.

Consequence

Example

$$\underbrace{A \vee B, A \rightarrow B, B \wedge R \rightarrow \neg A, R}_M \models (R \wedge \neg A) \wedge B$$

Proof:

Assume $\mathcal{A} \models F$ for all $F \in M$.

We need to prove $\mathcal{A} \models (R \wedge \neg A) \wedge B$.

It suffices to prove $\mathcal{A} \models R$, $\mathcal{A} \models \neg A$, and $\mathcal{A} \models B$

Consequence

Example

$$\underbrace{A \vee B, A \rightarrow B, B \wedge R \rightarrow \neg A, R}_M \models (R \wedge \neg A) \wedge B$$

Proof:

Assume $\mathcal{A} \models F$ for all $F \in M$.

We need to prove $\mathcal{A} \models (R \wedge \neg A) \wedge B$.

It suffices to prove $\mathcal{A} \models R$, $\mathcal{A} \models \neg A$, and $\mathcal{A} \models B$

- ▶ $\mathcal{A} \models R$ is immediate.

Consequence

Example

$$\underbrace{A \vee B, A \rightarrow B, B \wedge R \rightarrow \neg A, R}_M \models (R \wedge \neg A) \wedge B$$

Proof:

Assume $\mathcal{A} \models F$ for all $F \in M$.

We need to prove $\mathcal{A} \models (R \wedge \neg A) \wedge B$.

It suffices to prove $\mathcal{A} \models R$, $\mathcal{A} \models \neg A$, and $\mathcal{A} \models B$

- ▶ $\mathcal{A} \models R$ is immediate.
- ▶ $\mathcal{A} \models B$ follows from $\mathcal{A} \models A \vee B$ and $\mathcal{A} \models A \rightarrow B$:

Proof by cases:

If $\mathcal{A}(A) = 0$ then $\mathcal{A}(B) = 1$ because $\mathcal{A} \models A \vee B$

If $\mathcal{A}(A) = 1$ then $\mathcal{A}(B) = 1$ because $\mathcal{A} \models A \rightarrow B$

Consequence

Example

$$\underbrace{A \vee B, A \rightarrow B, B \wedge R \rightarrow \neg A, R}_M \models (R \wedge \neg A) \wedge B$$

Proof:

Assume $\mathcal{A} \models F$ for all $F \in M$.

We need to prove $\mathcal{A} \models (R \wedge \neg A) \wedge B$.

It suffices to prove $\mathcal{A} \models R$, $\mathcal{A} \models \neg A$, and $\mathcal{A} \models B$

- ▶ $\mathcal{A} \models R$ is immediate.
- ▶ $\mathcal{A} \models B$ follows from $\mathcal{A} \models A \vee B$ and $\mathcal{A} \models A \rightarrow B$:

Proof by cases:

If $\mathcal{A}(A) = 0$ then $\mathcal{A}(B) = 1$ because $\mathcal{A} \models A \vee B$

If $\mathcal{A}(A) = 1$ then $\mathcal{A}(B) = 1$ because $\mathcal{A} \models A \rightarrow B$

- ▶ $\mathcal{A} \models \neg A$ follows from $\mathcal{A} \models B$ and $\mathcal{A} \models R$.

Exercise

M	F	$M \models F?$
A	$A \vee B$	

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A		

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B		

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B		

Exercise

M	F	$M \models F ?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y
$A \wedge B$		

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y
$A \wedge B$	A	

Exercise

M	F	$M \models F ?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y
$A \wedge B$	A	Y

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y
$A \wedge B$	A	Y
$A \vee B$		

Exercise

M	F	$M \models F ?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y
$A \wedge B$	A	Y
$A \vee B$	A	

Exercise

M	F	$M \models F ?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y
$A \wedge B$	A	Y
$A \vee B$	A	N

Exercise

M	F	$M \models F?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y
$A \wedge B$	A	Y
$A \vee B$	A	N
$A, A \rightarrow B$		

Exercise

M	F	$M \models F ?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y
$A \wedge B$	A	Y
$A \vee B$	A	N
$A, A \rightarrow B$	B	

Exercise

M	F	$M \models F ?$
A	$A \vee B$	Y
A	$A \wedge B$	N
A, B	$A \vee B$	Y
A, B	$A \wedge B$	Y
$A \wedge B$	A	Y
$A \vee B$	A	N
$A, A \rightarrow B$	B	Y

Consequence

Exercise

The following statements are equivalent:

1. $F_1, \dots, F_k \models G$
2. $\models (\bigwedge_{i=1}^k F_i) \rightarrow G$

Proof of “if $F_1, \dots, F_k \models G$ then $\models \underbrace{(\bigwedge_{i=1}^k F_i) \rightarrow G}_H$ ”.

Assume $F_1, \dots, F_k \models G$.

We need to prove $\models H$, i.e. $\mathcal{A}(H) = 1$ for all \mathcal{A} .

Consequence

Exercise

The following statements are equivalent:

1. $F_1, \dots, F_k \models G$
2. $\models (\bigwedge_{i=1}^k F_i) \rightarrow G$

Proof of “if $F_1, \dots, F_k \models G$ then $\models \underbrace{(\bigwedge_{i=1}^k F_i) \rightarrow G}_H$ ”.

Assume $F_1, \dots, F_k \models G$.

We need to prove $\models H$, i.e. $\mathcal{A}(H) = 1$ for all \mathcal{A} .

We pick an arbitrary \mathcal{A} and show $\mathcal{A}(H) = 1$.

Proof by cases: either $\mathcal{A}(\bigwedge F_i) = 0$ or $\mathcal{A}(\bigwedge F_i) = 1$.

Consequence

Exercise

The following statements are equivalent:

1. $F_1, \dots, F_k \models G$
2. $\models (\bigwedge_{i=1}^k F_i) \rightarrow G$

Proof of “if $F_1, \dots, F_k \models G$ then $\models \underbrace{(\bigwedge_{i=1}^k F_i) \rightarrow G}_H$ ”.

Assume $F_1, \dots, F_k \models G$.

We need to prove $\models H$, i.e. $\mathcal{A}(H) = 1$ for all \mathcal{A} .

We pick an arbitrary \mathcal{A} and show $\mathcal{A}(H) = 1$.

Proof by cases: either $\mathcal{A}(\bigwedge F_i) = 0$ or $\mathcal{A}(\bigwedge F_i) = 1$.

- ▶ $\mathcal{A}(\bigwedge F_i) = 0$: Then $\mathcal{A}(H) = 1$ because $H = \bigwedge F_i \rightarrow G$.

Consequence

Exercise

The following statements are equivalent:

1. $F_1, \dots, F_k \models G$
2. $\models (\bigwedge_{i=1}^k F_i) \rightarrow G$

Proof of “if $F_1, \dots, F_k \models G$ then $\models \underbrace{(\bigwedge_{i=1}^k F_i) \rightarrow G}_H$ ”.

Assume $F_1, \dots, F_k \models G$.

We need to prove $\models H$, i.e. $\mathcal{A}(H) = 1$ for all \mathcal{A} .

We pick an arbitrary \mathcal{A} and show $\mathcal{A}(H) = 1$.

Proof by cases: either $\mathcal{A}(\bigwedge F_i) = 0$ or $\mathcal{A}(\bigwedge F_i) = 1$.

▶ $\mathcal{A}(\bigwedge F_i) = 0$: Then $\mathcal{A}(H) = 1$ because $H = \bigwedge F_i \rightarrow G$.

▶ $\mathcal{A}(\bigwedge F_i) = 1$: Then $\mathcal{A}(F_i) = 1$ for all i .

Therefore \mathcal{A} is a model of F_1, \dots, F_k .

Therefore $\mathcal{A} \models G$ because $F_1, \dots, F_k \models G$.

Consequence

Exercise

The following statements are equivalent:

1. $F_1, \dots, F_k \models G$
2. $\models (\bigwedge_{i=1}^k F_i) \rightarrow G$

Proof of “if $F_1, \dots, F_k \models G$ then $\models \underbrace{(\bigwedge_{i=1}^k F_i) \rightarrow G}_H$ ”.

Assume $F_1, \dots, F_k \models G$.

We need to prove $\models H$, i.e. $\mathcal{A}(H) = 1$ for all \mathcal{A} .

We pick an arbitrary \mathcal{A} and show $\mathcal{A}(H) = 1$.

Proof by cases: either $\mathcal{A}(\bigwedge F_i) = 0$ or $\mathcal{A}(\bigwedge F_i) = 1$.

▶ $\mathcal{A}(\bigwedge F_i) = 0$: Then $\mathcal{A}(H) = 1$ because $H = \bigwedge F_i \rightarrow G$.

▶ $\mathcal{A}(\bigwedge F_i) = 1$: Then $\mathcal{A}(F_i) = 1$ for all i .

Therefore \mathcal{A} is a model of F_1, \dots, F_k .

Therefore $\mathcal{A} \models G$ because $F_1, \dots, F_k \models G$.

Validity and satisfiability

Exercise

The following statements are equivalent:

1. $F \rightarrow G$ is valid.
2. $F \wedge \neg G$ is unsatisfiable.

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.		

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.	N	

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.	N	$\neg A \models A$

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.	N	$\neg A \models A$
If F valid then $M \models F$.		

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.	N	$\neg A \models A$
If F valid then $M \models F$.	Y	

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.	N	$\neg A \models A$
If F valid then $M \models F$.	Y	
If $F \in M$ then $M \models F$.		

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.	N	$\neg A \models A$
If F valid then $M \models F$.	Y	
If $F \in M$ then $M \models F$.	Y	

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.	N	$\neg A \models A$
If F valid then $M \models F$.	Y	
If $F \in M$ then $M \models F$.	Y	
If $F \models G$ then $\neg F \models \neg G$.		

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.	N	$\neg A \models A$
If F valid then $M \models F$.	Y	
If $F \in M$ then $M \models F$.	Y	
If $F \models G$ then $\neg F \models \neg G$.	N	

Exercise

Let M be a set of formulas, and let F and G be formulas.
Which of the following statements hold?

	Y/N	C.ex.
If F satisfiable then $M \models F$.	N	$\neg A \models A$
If F valid then $M \models F$.	Y	
If $F \in M$ then $M \models F$.	Y	
If $F \models G$ then $\neg F \models \neg G$.	N	$A \models A \vee B$

Notation

Warning: The symbol \models is overloaded:

$$\mathcal{A} \models F$$

$$\models F$$

$$M \models F$$

Convenient variations for set of formulas S :

$$\mathcal{A} \models S \text{ means that for all } F \in S, \mathcal{A} \models F$$

$$\models S \text{ means that for all } F \in S, \models F$$

$$M \models S \text{ means that for all } F \in S, M \models F$$

Propositional Logic Equivalences

Equivalence

Definition (Equivalence)

Two formulas F and G are (semantically) equivalent if $\mathcal{A}(F) = \mathcal{A}(G)$ for every assignment \mathcal{A} .

We write $F \equiv G$ to denote that F and G are equivalent.

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Exercise

Which of the following equivalences hold?

$$(A \wedge (A \vee B)) \equiv A$$

$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \rightarrow B) \rightarrow C)$$

$$(A \rightarrow (B \rightarrow C)) \equiv ((A \wedge B) \rightarrow C)$$

$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(A \rightarrow B) \equiv (\neg A \rightarrow \neg B)$$

$$(A \leftrightarrow (B \leftrightarrow C)) \equiv ((A \leftrightarrow B) \leftrightarrow C)$$

Observation

The following connections hold:

$$\begin{aligned} \models F \rightarrow G & \text{ iff } F \models G \\ \models F \leftrightarrow G & \text{ iff } F \equiv G \end{aligned}$$

NB: “iff” means “if and only if”

Reductions between problems (I)

- ▶ **Validity** to **Unsatisfiability** (and back):

F valid iff $\neg F$ unsatisfiable

F unsatisfiable iff $\neg F$ valid

- ▶ **Validity** to **Consequence**:

F valid iff $\top \models F$

- ▶ **Consequence** to **Validity**:

$F \models G$ iff $F \rightarrow G$ valid

Reductions between problems (II)

- ▶ **Validity** to **Equivalence**:

$$F \text{ valid} \quad \text{iff} \quad F \equiv \top$$

- ▶ **Equivalence** to **Validity**:

$$F \equiv G \quad \text{iff} \quad F \leftrightarrow G \text{ valid}$$

Properties of semantic equivalence

- ▶ Semantic equivalence is an **equivalence relation** between formulas.
- ▶ Semantic equivalence is **closed under operators**:

If $F_1 \equiv F_2$ and $G_1 \equiv G_2$

then $\neg F_1 \equiv \neg F_2$ and

$(F_1 \circ G_1) \equiv (F_2 \circ G_2)$ for $\circ \in \{\vee, \wedge, \rightarrow, \iff\}$

Equivalence relation + Closure under Operations

=

Congruence relation

Replacement theorem

Theorem

Let $F \equiv G$. Let H be a formula with an occurrence of F as a subformula. Let H' be the result of replacing an arbitrary occurrence of F in H by G . Then $H \equiv H'$.

Replacement theorem

Theorem

Let $F \equiv G$. Let H be a formula with an occurrence of F as a subformula. Let H' be the result of replacing an arbitrary occurrence of F in H by G . Then $H \equiv H'$.

Proof by induction on the structure of H .

We consider only the case $H = \neg H_0$.

Replacement theorem

Theorem

Let $F \equiv G$. Let H be a formula with an occurrence of F as a subformula. Let H' be the result of replacing an arbitrary occurrence of F in H by G . Then $H \equiv H'$.

Proof by induction on the structure of H .

We consider only the case $H = \neg H_0$.

Two cases: either $F = H$ or F is a subformula of H_0 .

Replacement theorem

Theorem

Let $F \equiv G$. Let H be a formula with an occurrence of F as a subformula. Let H' be the result of replacing an arbitrary occurrence of F in H by G . Then $H \equiv H'$.

Proof by induction on the structure of H .

We consider only the case $H = \neg H_0$.

Two cases: either $F = H$ or F is a subformula of H_0 .

- ▶ $F = H$: Then $H' = G$ and thus $H = F \equiv G = H'$.

Replacement theorem

Theorem

Let $F \equiv G$. Let H be a formula with an occurrence of F as a subformula. Let H' be the result of replacing an arbitrary occurrence of F in H by G . Then $H \equiv H'$.

Proof by induction on the structure of H .

We consider only the case $H = \neg H_0$.

Two cases: either $F = H$ or F is a subformula of H_0 .

- ▶ $F = H$: Then $H' = G$ and thus $H = F \equiv G = H'$.
- ▶ F is a subformula of H_0 .
Let H'_0 be the result of replacing F by G in H_0 .

Replacement theorem

Theorem

Let $F \equiv G$. Let H be a formula with an occurrence of F as a subformula. Let H' be the result of replacing an arbitrary occurrence of F in H by G . Then $H \equiv H'$.

Proof by induction on the structure of H .

We consider only the case $H = \neg H_0$.

Two cases: either $F = H$ or F is a subformula of H_0 .

- ▶ $F = H$: Then $H' = G$ and thus $H = F \equiv G = H'$.
- ▶ F is a subformula of H_0 .

Let H'_0 be the result of replacing F by G in H_0 .

IH: $H_0 \equiv H'_0$

Replacement theorem

Theorem

Let $F \equiv G$. Let H be a formula with an occurrence of F as a subformula. Let H' be the result of replacing an arbitrary occurrence of F in H by G . Then $H \equiv H'$.

Proof by induction on the structure of H .

We consider only the case $H = \neg H_0$.

Two cases: either $F = H$ or F is a subformula of H_0 .

- ▶ $F = H$: Then $H' = G$ and thus $H = F \equiv G = H'$.
- ▶ F is a subformula of H_0 .

Let H'_0 be the result of replacing F by G in H_0 .

IH: $H_0 \equiv H'_0$

Thus $H = \neg H_0 \equiv \neg H'_0 = H'$.

Equivalences (I)

Theorem

$$(F \wedge F) \equiv F$$

$$(F \vee F) \equiv F$$

$$(F \wedge G) \equiv (G \wedge F)$$

$$(F \vee G) \equiv (G \vee F)$$

$$((F \wedge G) \wedge H) \equiv (F \wedge (G \wedge H))$$

$$((F \vee G) \vee H) \equiv (F \vee (G \vee H))$$

$$(F \wedge (F \vee G)) \equiv F$$

$$(F \vee (F \wedge G)) \equiv F$$

(Idempotence)

(Commutativity)

(Associativity)

(Absorption)

Equivalences (II)

$$(F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H))$$

$$(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H))$$

(Distributivity)

$$\neg\neg F \equiv F$$

(Double negation)

$$\neg(F \wedge G) \equiv (\neg F \vee \neg G)$$

$$\neg(F \vee G) \equiv (\neg F \wedge \neg G)$$

(deMorgan's Laws)

$$\neg\top \equiv \perp$$

$$\neg\perp \equiv \top$$

$$(\top \vee G) \equiv \top$$

$$(\top \wedge G) \equiv G$$

$$(\perp \vee G) \equiv G$$

$$(\perp \wedge G) \equiv \perp$$

Warning

The symbols \models and \equiv are **not** operators
in the language of propositional logic
but part of the meta-language for talking about logic.

Examples:

$\mathcal{A} \models F$ and $F \equiv G$ are not propositional formulas.
 $(\mathcal{A} \models F) \equiv G$ and $(F \equiv G) \leftrightarrow (G \equiv F)$ are nonsense.

Propositional Logic

Normal Forms

Abbreviations

Until further notice:

$F_1 \rightarrow F_2$ abbreviates $\neg F_1 \vee F_2$

$F_1 \leftrightarrow F_2$ abbreviates $(F_1 \wedge F_2) \vee (\neg F_1 \wedge \neg F_2)$

\top abbreviates $A_1 \vee \neg A_1$

\perp abbreviates $A_1 \wedge \neg A_1$

Literals

Definition

A **literal** is an atom or the negation of an atom.
In the former case the literal is **positive**,
in the latter case it is **negative**.

Negation Normal Form (NNF)

Definition

A formula is in **negation normal form (NNF)** if negation (\neg) occurs only directly in front of atoms.

Example

In NNF: $\neg A \wedge \neg B$

Not in NNF: $\neg(A \vee B)$

Transformation into NNF

Any formula can be transformed into an equivalent formula in NNF by pushing \neg inwards. Apply the following equivalences from left to right as long as possible:

$$\begin{aligned}\neg\neg F &\equiv F \\ \neg(F \wedge G) &\equiv (\neg F \vee \neg G) \\ \neg(F \vee G) &\equiv (\neg F \wedge \neg G)\end{aligned}$$

Transformation into NNF

Any formula can be transformed into an equivalent formula in NNF by pushing \neg inwards. Apply the following equivalences from left to right as long as possible:

$$\begin{aligned}\neg\neg F &\equiv F \\ \neg(F \wedge G) &\equiv (\neg F \vee \neg G) \\ \neg(F \vee G) &\equiv (\neg F \wedge \neg G)\end{aligned}$$

Example

$$\begin{aligned}(\neg(A \wedge \neg B) \wedge C) &\equiv ((\neg A \vee \neg\neg B) \wedge C) \equiv ((\neg A \vee B) \wedge C) \\ (\text{"}F \equiv G \equiv H\text{" is an abbreviation for "}F \equiv G \text{ and } G \equiv H\text{"})\end{aligned}$$

Transformation into NNF

Any formula can be transformed into an equivalent formula in NNF by pushing \neg inwards. Apply the following equivalences from left to right as long as possible:

$$\begin{aligned}\neg\neg F &\equiv F \\ \neg(F \wedge G) &\equiv (\neg F \vee \neg G) \\ \neg(F \vee G) &\equiv (\neg F \wedge \neg G)\end{aligned}$$

Example

$$\begin{aligned}(\neg(A \wedge \neg B) \wedge C) &\equiv ((\neg A \vee \neg\neg B) \wedge C) \equiv ((\neg A \vee B) \wedge C) \\ (\text{"}F \equiv G \equiv H\text{" is an abbreviation for "}F \equiv G \text{ and } G \equiv H\text{"})\end{aligned}$$

Does this process always terminate? Is the result unique?

CNF and DNF

Definition

A formula F is in **conjunctive normal form (CNF)** if it is a conjunction of disjunctions of literals:

$$F = \left(\bigwedge_{i=1}^n \left(\bigvee_{j=1}^{m_i} L_{i,j} \right) \right),$$

where $L_{i,j} \in \{A_1, A_2, \dots\} \cup \{\neg A_1, \neg A_2, \dots\}$

Definition

A formula F is in **disjunctive normal form (DNF)** if it is a disjunction of conjunctions of literals:

$$F = \left(\bigvee_{i=1}^n \left(\bigwedge_{j=1}^{m_i} L_{i,j} \right) \right),$$

where $L_{i,j} \in \{A_1, A_2, \dots\} \cup \{\neg A_1, \neg A_2, \dots\}$

Transformation into CNF and DNF

Any formula can be transformed into an equivalent formula in CNF or DNF in two steps:

1. Transform the initial formula into its NNF
2. Transform the NNF into CNF or DNF:
 - ▶ Transformation into CNF. Apply the following equivalences from left to right as long as possible:

$$(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H))$$

$$((F \wedge G) \vee H) \equiv ((F \vee H) \wedge (G \vee H))$$

- ▶ Transformation into DNF. Apply the following equivalences from left to right as long as possible:

$$(F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H))$$

$$((F \vee G) \wedge H) \equiv ((F \wedge H) \vee (G \wedge H))$$

Transformation into CNF and DNF

Any formula can be transformed into an equivalent formula in CNF or DNF in two steps:

1. Transform the initial formula into its NNF
2. Transform the NNF into CNF or DNF:
 - ▶ Transformation into CNF. Apply the following equivalences from left to right as long as possible:

$$(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H))$$

$$((F \wedge G) \vee H) \equiv ((F \vee H) \wedge (G \vee H))$$

- ▶ Transformation into DNF. Apply the following equivalences from left to right as long as possible:

$$(F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H))$$

$$((F \vee G) \wedge H) \equiv ((F \wedge H) \vee (G \wedge H))$$

Termination

Why does the transformation into NNF and CNF terminate?

Termination

Why does the transformation into NNF and CNF terminate?

Challenge Question: Find a weight function $w :: \text{formula} \rightarrow \mathbb{N}$ such that $w(l.h.s.) > w(r.h.s.)$ for the equivalences

$$\begin{aligned}\neg\neg F &\equiv F \\ \neg(F \wedge G) &\equiv (\neg F \vee \neg G) \\ \neg(F \vee G) &\equiv (\neg F \wedge \neg G) \\ (F \vee (G \wedge H)) &\equiv ((F \vee G) \wedge (F \vee H)) \\ ((F \wedge G) \vee H) &\equiv ((F \vee H) \wedge (G \vee H))\end{aligned}$$

Define w recursively:

$$w(A_i) = \dots$$

$$w(\neg F) = \dots w(F) \dots$$

$$w(F \wedge G) = \dots w(F) \dots w(G) \dots$$

$$w(F \vee G) = \dots w(F) \dots w(G) \dots$$

Complexity considerations

The CNF and DNF of a formula of size n can have size 2^n

Can we do better? Yes, if we do not insist on \equiv .

Definition

Two formulas F and G are **equisatisfiable** if F is satisfiable iff G is satisfiable.

Theorem

For every formula F of size n there is an equisatisfiable CNF formula G of size $O(n)$.

Propositional Logic
Definitional CNF
(Tseytin's transformation)

Definitional CNF

The **definitional CNF** of a formula is obtained in 2 steps:

1. Repeatedly replace a subformula G of the form $\neg A'$, $A' \wedge B'$ or $A' \vee B'$ (where A', B' atoms!) by a new atom A and conjoin $A \leftrightarrow G$.

This replacement is not applied to the “definitions” $A \leftrightarrow G$ but only to the (remains of the) original formula.

2. Translate all the subformulas $A \leftrightarrow G$ into CNF.

Definitional CNF

The **definitional CNF** of a formula is obtained in 2 steps:

1. Repeatedly replace a subformula G of the form $\neg A'$, $A' \wedge B'$ or $A' \vee B'$ (where A', B' atoms!) by a new atom A and conjoin $A \leftrightarrow G$.

This replacement is not applied to the “definitions” $A \leftrightarrow G$ but only to the (remains of the) original formula.

2. Translate all the subformulas $A \leftrightarrow G$ into CNF.

Example

$$\neg(A_1 \vee A_2) \wedge A_3$$

\rightsquigarrow

$$\neg A_4 \wedge A_3 \wedge (A_4 \leftrightarrow (A_1 \vee A_2))$$

\rightsquigarrow

$$A_5 \wedge A_3 \wedge (A_4 \leftrightarrow (A_1 \vee A_2)) \wedge (A_5 \leftrightarrow \neg A_4)$$

\rightsquigarrow

$$A_5 \wedge A_3 \wedge \text{CNF}(A_4 \leftrightarrow (A_1 \vee A_2)) \wedge \text{CNF}(A_5 \leftrightarrow \neg A_4)$$

Definitional CNF: Complexity

Let the initial formula have size n .

1. Each replacement step increases the size of the formula by a constant.
There are at most as many replacement steps as subformulas, linearly many.
2. The conversion of each $A \leftrightarrow G$ into CNF increases the size by a constant.
There are only linearly many such subformulas.

Thus the definitional CNF has size $O(n)$.

Definitional CNF: Correctness — Notation

Definition

The notation $F[G/A]$ denotes the result of replacing **all** occurrences of the atom A in F by G .

We pronounce it as “ F with G for A ”.

Example

$$(A \wedge B)[(A \rightarrow B)/B] = (A \wedge (A \rightarrow B))$$

Definition

The notation $\mathcal{A}[v/A]$ denotes a modified version of \mathcal{A} that maps A to v and behaves like \mathcal{A} otherwise:

$$(\mathcal{A}[v/A])(A_i) = \begin{cases} v & \text{if } A_i = A \\ \mathcal{A}(A_i) & \text{otherwise} \end{cases}$$

Definitional CNF: Correctness — Substitution Lemma

Lemma

$$\mathcal{A}(F[G/A]) = \mathcal{A}'(F) \text{ where } \mathcal{A}' = \mathcal{A}[\mathcal{A}(G)/A]$$

Definitional CNF: Correctness — Substitution Lemma

Lemma

$\mathcal{A}(F[G/A]) = \mathcal{A}'(F)$ where $\mathcal{A}' = \mathcal{A}[\mathcal{A}(G)/A]$

Proof by structural induction on F .

▶ F is an atom:

If $F = A$: $\mathcal{A}(F[G/A]) = \mathcal{A}(G) = \mathcal{A}'(F)$

If $F \neq A$: $\mathcal{A}(F[G/A]) = \mathcal{A}(F) = \mathcal{A}'(F)$

▶ $F = F_1 \wedge F_2$:

$$\begin{aligned}\mathcal{A}((F_1 \wedge F_2)[G/A]) &= \mathcal{A}(F_1[G/A] \wedge F_2[G/A]) \\ &= \min(\mathcal{A}(F_1[G/A]), \mathcal{A}(F_2[G/A])) \\ &\stackrel{IH}{=} \min(\mathcal{A}'(F_1), \mathcal{A}'(F_2)) \\ &= \mathcal{A}'(F_1 \wedge F_2)\end{aligned}$$

Definitional CNF: Correctness

Each replacement step produces an equisatisfiable formula:

Lemma

Let A be an atom that does not occur in G .

Then $F[G/A]$ is equisatisfiable with $F \wedge (A \leftrightarrow G)$.

Definitional CNF: Correctness

Each replacement step produces an equisatisfiable formula:

Lemma

Let A be an atom that does not occur in G .

Then $F[G/A]$ is equisatisfiable with $F \wedge (A \leftrightarrow G)$.

Proof Assume $\mathcal{A} \models F[G/A]$ for some assignment \mathcal{A} .

By the Substitution Lemma, $\mathcal{A}' = \mathcal{A}[\mathcal{A}(G)/A]$ is a model of F .

Moreover $\mathcal{A}' \models (A \leftrightarrow G)$ because $\mathcal{A}'(A) = \mathcal{A}(G)$ and

$\mathcal{A}(G) = \mathcal{A}'(G)$ because A does not appear in G .

Thus $\mathcal{A}' \models F \wedge (A \leftrightarrow G)$ and so $F \wedge (A \leftrightarrow G)$ is satisfiable.

Definitional CNF: Correctness

Each replacement step produces an equisatisfiable formula:

Lemma

Let A be an atom that does not occur in G .

Then $F[G/A]$ is equisatisfiable with $F \wedge (A \leftrightarrow G)$.

Proof Assume $\mathcal{A} \models F[G/A]$ for some assignment \mathcal{A} .

By the Substitution Lemma, $\mathcal{A}' = \mathcal{A}[\mathcal{A}(G)/A]$ is a model of F .

Moreover $\mathcal{A}' \models (A \leftrightarrow G)$ because $\mathcal{A}'(A) = \mathcal{A}(G)$ and

$\mathcal{A}(G) = \mathcal{A}'(G)$ because A does not appear in G .

Thus $\mathcal{A}' \models F \wedge (A \leftrightarrow G)$ and so $F \wedge (A \leftrightarrow G)$ is satisfiable.

Assume $\mathcal{A} \models F \wedge (A \leftrightarrow G)$ for some assignment \mathcal{A} .

This implies $\mathcal{A}(A) = \mathcal{A}(G)$ and so $\mathcal{A}[\mathcal{A}(G)/A] = \mathcal{A}$.

Thus $\mathcal{A}(F[G/A]) = (\mathcal{A}[\mathcal{A}(G)/A])(F) = \mathcal{A}(F) = 1$ by the Substitution Lemma.

Definitional CNF: Correctness

Each replacement step produces an equisatisfiable formula:

Lemma

Let A be an atom that does not occur in G .

Then $F[G/A]$ is equisatisfiable with $F \wedge (A \leftrightarrow G)$.

Proof Assume $\mathcal{A} \models F[G/A]$ form some assignment \mathcal{A} .

By the Substitution Lemma, $\mathcal{A}' = \mathcal{A}[\mathcal{A}(G)/A]$ is a model of F .

Moreover $\mathcal{A}' \models (A \leftrightarrow G)$ because $\mathcal{A}'(A) = \mathcal{A}(G)$ and

$\mathcal{A}(G) = \mathcal{A}'(G)$ because A does not appear in G .

Thus $\mathcal{A}' \models F \wedge (A \leftrightarrow G)$ and so $F \wedge (A \leftrightarrow G)$ is satisfiable.

Assume $\mathcal{A} \models F \wedge (A \leftrightarrow G)$ form some assignment \mathcal{A} .

This implies $\mathcal{A}(A) = \mathcal{A}(G)$ and so $\mathcal{A}[\mathcal{A}(G)/A] = \mathcal{A}$.

Thus $\mathcal{A}(F[G/A]) = (\mathcal{A}[\mathcal{A}(G)/A])(F) = \mathcal{A}(F) = 1$ by the Substitution Lemma.

Does $F \wedge (A \leftrightarrow G) \models F[G/A]$ hold? Does
 $F[G/A] \models F \wedge (A \leftrightarrow G)$ hold?

Summary

Theorem

*For every formula F of size n
there is an **equisatisfiable CNF** formula G of size $O(n)$.*

Similarly it can be shown:

Theorem

*For every formula F of size n
there is an **equivalent DNF** formula G of size $O(n)$.*

Validity of CNF

Validity of formulas in CNF can be checked in linear time.

A formula in CNF is valid iff all its disjunctions are valid.

A disjunction is valid iff it contains both an atomic A and $\neg A$ as literals.

Example

Valid: $(A \vee \neg A \vee B) \wedge (C \vee \neg C)$

Not valid: $(A \vee \neg A) \wedge (\neg A \vee C)$

Satisfiability of DNF

Satisfiability of formulas in DNF can be checked in linear time.

A formula in DNF is satisfiable iff at least one of its conjunctions is satisfiable. A conjunction is satisfiable iff it does not contain both an atomic A and $\neg A$ as literals.

Example

Satisfiable: $(\neg B \wedge A \wedge B) \vee (\neg A \wedge C)$

Unsatisfiable: $(A \wedge \neg A \wedge B) \vee (C \wedge \neg C)$

Satisfiability/validity of DNF and CNF

Theorem

Satisfiability of formulas in CNF is NP-complete.

Theorem

Validity of formulas in DNF is co-NP-complete.

Standard decision procedure for validity of F :

1. Transform $\neg F$ into an equisat. formula G in def. CNF
2. Apply efficient CNF-based SAT solver to G

Propositional Logic

Horn Formulas

Efficient satisfiability checks

In the following:

- ▶ A very efficient satisfiability check for the special class of **Horn formulas**.

Efficient satisfiability checks

In the following:

- ▶ A very efficient satisfiability check for the special class of **Horn formulas**.
- ▶ Efficient satisfiability checks for arbitrary formulas in CNF: **resolution** (later).

Horn formulas

Definition

A formula F in CNF is a **Horn formula** if every disjunction in F contains at most one positive literal.

Horn formulas

Definition

A formula F in CNF is a **Horn formula** if every disjunction in F contains at most one positive literal.

A disjunction in a Horn formula can equivalently be viewed as an implication $K \rightarrow B$ where K is a conjunction of atoms or $K = \top$ and B is an atom or $B = \perp$:

Horn formulas

Definition

A formula F in CNF is a **Horn formula** if every disjunction in F contains at most one positive literal.

A disjunction in a Horn formula can equivalently be viewed as an implication $K \rightarrow B$ where K is a conjunction of atoms or $K = \top$ and B is an atom or $B = \perp$:

$$(\neg A \vee \neg B \vee C) \equiv (A \wedge B \rightarrow C)$$

Horn formulas

Definition

A formula F in CNF is a **Horn formula** if every disjunction in F contains at most one positive literal.

A disjunction in a Horn formula can equivalently be viewed as an implication $K \rightarrow B$ where K is a conjunction of atoms or $K = \top$ and B is an atom or $B = \perp$:

$$\begin{aligned}(\neg A \vee \neg B \vee C) &\equiv (A \wedge B \rightarrow C) \\ (\neg A \vee \neg B) &\equiv\end{aligned}$$

Horn formulas

Definition

A formula F in CNF is a **Horn formula** if every disjunction in F contains at most one positive literal.

A disjunction in a Horn formula can equivalently be viewed as an implication $K \rightarrow B$ where K is a conjunction of atoms or $K = \top$ and B is an atom or $B = \perp$:

$$\begin{aligned}(\neg A \vee \neg B \vee C) &\equiv (A \wedge B \rightarrow C) \\(\neg A \vee \neg B) &\equiv (A \wedge B \rightarrow \perp)\end{aligned}$$

Horn formulas

Definition

A formula F in CNF is a **Horn formula** if every disjunction in F contains at most one positive literal.

A disjunction in a Horn formula can equivalently be viewed as an implication $K \rightarrow B$ where K is a conjunction of atoms or $K = \top$ and B is an atom or $B = \perp$:

$$\begin{aligned}(\neg A \vee \neg B \vee C) &\equiv (A \wedge B \rightarrow C) \\(\neg A \vee \neg B) &\equiv (A \wedge B \rightarrow \perp) \\A &\equiv\end{aligned}$$

Horn formulas

Definition

A formula F in CNF is a **Horn formula** if every disjunction in F contains at most one positive literal.

A disjunction in a Horn formula can equivalently be viewed as an implication $K \rightarrow B$ where K is a conjunction of atoms or $K = \top$ and B is an atom or $B = \perp$:

$$\begin{aligned}(\neg A \vee \neg B \vee C) &\equiv (A \wedge B \rightarrow C) \\(\neg A \vee \neg B) &\equiv (A \wedge B \rightarrow \perp) \\A &\equiv (\top \rightarrow A)\end{aligned}$$

Satisfiability check for Horn formulas

Input: a Horn formula F .

Satisfiability check for Horn formulas

Input: a Horn formula F .

Algorithm building a model (assignment) \mathcal{M} :

Satisfiability check for Horn formulas

Input: a Horn formula F .

Algorithm building a model (assignment) \mathcal{M} :

for all atoms A_i in F **do** $\mathcal{M}(A_i) := 0$;

Satisfiability check for Horn formulas

Input: a Horn formula F .

Algorithm building a model (assignment) \mathcal{M} :

for all atoms A_i in F **do** $\mathcal{M}(A_i) := 0$;

while F has a subformula $K \rightarrow B$
such that $\mathcal{M}(K) = 1$ and $\mathcal{M}(B) = 0$

do

Satisfiability check for Horn formulas

Input: a Horn formula F .

Algorithm building a model (assignment) \mathcal{M} :

for all atoms A_i in F **do** $\mathcal{M}(A_i) := 0$;

while F has a subformula $K \rightarrow B$
such that $\mathcal{M}(K) = 1$ and $\mathcal{M}(B) = 0$

do

if $B = \perp$ **then return** “unsatisfiable”

else $\mathcal{M}(B) := 1$

Satisfiability check for Horn formulas

Input: a Horn formula F .

Algorithm building a model (assignment) \mathcal{M} :

for all atoms A_i in F **do** $\mathcal{M}(A_i) := 0$;

while F has a subformula $K \rightarrow B$
such that $\mathcal{M}(K) = 1$ and $\mathcal{M}(B) = 0$

do

if $B = \perp$ **then return** “unsatisfiable”

else $\mathcal{M}(B) := 1$

return “satisfiable”

Satisfiability check for Horn formulas

Input: a Horn formula F .

Algorithm building a model (assignment) \mathcal{M} :

```
for all atoms  $A_i$  in  $F$  do  $\mathcal{M}(A_i) := 0$ ;  
while  $F$  has a subformula  $K \rightarrow B$   
      such that  $\mathcal{M}(K) = 1$  and  $\mathcal{M}(B) = 0$   
do  
      if  $B = \perp$  then return “unsatisfiable”  
      else  $\mathcal{M}(B) := 1$   
return “satisfiable”
```

Maximal number of iterations of the while loop:
number of implications in F

Satisfiability check for Horn formulas

Input: a Horn formula F .

Algorithm building a model (assignment) \mathcal{M} :

```
for all atoms  $A_i$  in  $F$  do  $\mathcal{M}(A_i) := 0$ ;  
while  $F$  has a subformula  $K \rightarrow B$   
    such that  $\mathcal{M}(K) = 1$  and  $\mathcal{M}(B) = 0$   
do  
    if  $B = \perp$  then return “unsatisfiable”  
    else  $\mathcal{M}(B) := 1$   
return “satisfiable”
```

Maximal number of iterations of the while loop:
number of implications in F

Each iteration requires at most $O(|F|)$ steps.

Satisfiability check for Horn formulas

Input: a Horn formula F .

Algorithm building a model (assignment) \mathcal{M} :

```
for all atoms  $A_i$  in  $F$  do  $\mathcal{M}(A_i) := 0$ ;  
while  $F$  has a subformula  $K \rightarrow B$   
      such that  $\mathcal{M}(K) = 1$  and  $\mathcal{M}(B) = 0$   
do  
  if  $B = \perp$  then return “unsatisfiable”  
  else  $\mathcal{M}(B) := 1$   
return “satisfiable”
```

Maximal number of iterations of the while loop:
 number of implications in F

Each iteration requires at most $O(|F|)$ steps.

Overall complexity: $O(|F|^2)$

Satisfiability check for Horn formulas

Input: a Horn formula F .

Algorithm building a model (assignment) \mathcal{M} :

```
for all atoms  $A_i$  in  $F$  do  $\mathcal{M}(A_i) := 0$ ;  
while  $F$  has a subformula  $K \rightarrow B$   
      such that  $\mathcal{M}(K) = 1$  and  $\mathcal{M}(B) = 0$   
do  
  if  $B = \perp$  then return “unsatisfiable”  
  else  $\mathcal{M}(B) := 1$   
return “satisfiable”
```

Maximal number of iterations of the while loop:
 number of implications in F

Each iteration requires at most $O(|F|)$ steps.

Overall complexity: $O(|F|^2)$

[Algorithm can be improved to $O(|F|)$. See Schönig.]

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Proof Observe: if the algorithm sets $\mathcal{M}(B) = 1$, then $\mathcal{A}(B) = 1$ for every assignment \mathcal{A} such that $\mathcal{A}(F) = 1$.

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Proof Observe: if the algorithm sets $\mathcal{M}(B) = 1$, then $\mathcal{A}(B) = 1$ for every assignment \mathcal{A} such that $\mathcal{A}(F) = 1$. This is an invariant.

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Proof Observe: if the algorithm sets $\mathcal{M}(B) = 1$, then $\mathcal{A}(B) = 1$ for every assignment \mathcal{A} such that $\mathcal{A}(F) = 1$. This is an invariant.

(a) If “unsatisfiable” then unsatisfiable.

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Proof Observe: if the algorithm sets $\mathcal{M}(B) = 1$, then $\mathcal{A}(B) = 1$ for every assignment \mathcal{A} such that $\mathcal{A}(F) = 1$. This is an invariant.

(a) If “unsatisfiable” then unsatisfiable.

We prove unsatisfiability by contradiction.

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Proof Observe: if the algorithm sets $\mathcal{M}(B) = 1$, then $\mathcal{A}(B) = 1$ for every assignment \mathcal{A} such that $\mathcal{A}(F) = 1$. This is an invariant.

(a) If “unsatisfiable” then unsatisfiable.

We prove unsatisfiability by contradiction.

Assume $\mathcal{A}(F) = 1$ for some \mathcal{A} .

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Proof Observe: if the algorithm sets $\mathcal{M}(B) = 1$, then $\mathcal{A}(B) = 1$ for every assignment \mathcal{A} such that $\mathcal{A}(F) = 1$. This is an invariant.

(a) If “unsatisfiable” then unsatisfiable.

We prove unsatisfiability by contradiction.

Assume $\mathcal{A}(F) = 1$ for some \mathcal{A} .

Let $(A_{i_1} \wedge \dots \wedge A_{i_k} \rightarrow \perp)$ be the subformula causing “unsatisfiable”.

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Proof Observe: if the algorithm sets $\mathcal{M}(B) = 1$, then $\mathcal{A}(B) = 1$ for every assignment \mathcal{A} such that $\mathcal{A}(F) = 1$. This is an invariant.

(a) If “unsatisfiable” then unsatisfiable.

We prove unsatisfiability by contradiction.

Assume $\mathcal{A}(F) = 1$ for some \mathcal{A} .

Let $(A_{i_1} \wedge \dots \wedge A_{i_k} \rightarrow \perp)$ be the subformula causing “unsatisfiable”.

Since $\mathcal{M}(A_{i_1}) = \dots = \mathcal{M}(A_{i_k}) = 1$, $\mathcal{A}(A_{i_1}) = \dots = \mathcal{A}(A_{i_k}) = 1$.

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Proof Observe: if the algorithm sets $\mathcal{M}(B) = 1$, then $\mathcal{A}(B) = 1$ for every assignment \mathcal{A} such that $\mathcal{A}(F) = 1$. This is an invariant.

(a) If “unsatisfiable” then unsatisfiable.

We prove unsatisfiability by contradiction.

Assume $\mathcal{A}(F) = 1$ for some \mathcal{A} .

Let $(A_{i_1} \wedge \dots \wedge A_{i_k} \rightarrow \perp)$ be the subformula causing “unsatisfiable”.

Since $\mathcal{M}(A_{i_1}) = \dots = \mathcal{M}(A_{i_k}) = 1$, $\mathcal{A}(A_{i_1}) = \dots = \mathcal{A}(A_{i_k}) = 1$.

Then $\mathcal{A}(A_{i_1} \wedge \dots \wedge A_{i_k} \rightarrow \perp) = 0$ and so $\mathcal{A}(F) = 0$, contradiction.

Correctness of the model building algorithm

Theorem

The algorithm returns “satisfiable” iff F is satisfiable.

Proof Observe: if the algorithm sets $\mathcal{M}(B) = 1$, then $\mathcal{A}(B) = 1$ for every assignment \mathcal{A} such that $\mathcal{A}(F) = 1$. This is an invariant.

(a) If “unsatisfiable” then unsatisfiable.

We prove unsatisfiability by contradiction.

Assume $\mathcal{A}(F) = 1$ for some \mathcal{A} .

Let $(A_{i_1} \wedge \dots \wedge A_{i_k} \rightarrow \perp)$ be the subformula causing “unsatisfiable”.

Since $\mathcal{M}(A_{i_1}) = \dots = \mathcal{M}(A_{i_k}) = 1$, $\mathcal{A}(A_{i_1}) = \dots = \mathcal{A}(A_{i_k}) = 1$.

Then $\mathcal{A}(A_{i_1} \wedge \dots \wedge A_{i_k} \rightarrow \perp) = 0$ and so $\mathcal{A}(F) = 0$, contradiction.

So F has no satisfying assignments.

(b) If “satisfiable” then satisfiable.

(b) If “satisfiable” then satisfiable.

After termination with “satisfiable”,

for every subformula $K \rightarrow B$ of F , $\mathcal{M}(K) = 0$ or $\mathcal{M}(B) = 1$.

(b) If “satisfiable” then satisfiable.

After termination with “satisfiable”,

for every subformula $K \rightarrow B$ of F , $\mathcal{M}(K) = 0$ or $\mathcal{M}(B) = 1$.

Therefore $\mathcal{M}(K \rightarrow B) = 1$ and thus $\mathcal{M} \models F$.

(b) If “satisfiable” then satisfiable.

After termination with “satisfiable”,

for every subformula $K \rightarrow B$ of F , $\mathcal{M}(K) = 0$ or $\mathcal{M}(B) = 1$.

Therefore $\mathcal{M}(K \rightarrow B) = 1$ and thus $\mathcal{M} \models F$.

In fact, the invariant shows that \mathcal{M} is the **minimal** model of F .

Propositional Logic

Compactness

Compactness Theorem

Theorem

*A set S of formulas is satisfiable
iff every finite subset of S is satisfiable.*

Compactness Theorem

Theorem

*A set S of formulas is satisfiable
iff every finite subset of S is satisfiable.*

Equivalent formulation:

*A set S of formulas is unsatisfiable
iff some finite subset of S is unsatisfiable.*

An application: Graph Coloring

Definition

A **4-coloring** of a graph (V, E) is a map $c : V \rightarrow \{1, 2, 3, 4\}$ such that $(x, y) \in E$ implies $c(x) \neq c(y)$.

An application: Graph Coloring

Definition

A **4-coloring** of a graph (V, E) is a map $c : V \rightarrow \{1, 2, 3, 4\}$ such that $(x, y) \in E$ implies $c(x) \neq c(y)$.

Theorem (4CT)

An finite planar graph has a 4-coloring.

An application: Graph Coloring

Definition

A **4-coloring** of a graph (V, E) is a map $c : V \rightarrow \{1, 2, 3, 4\}$ such that $(x, y) \in E$ implies $c(x) \neq c(y)$.

Theorem (4CT)

An finite planar graph has a 4-coloring.

Theorem

A planar graph $G = (V, E)$ with countably many vertices $V = \{v_1, v_2, \dots\}$ has a 4-coloring.

An application: Graph Coloring

Definition

A **4-coloring** of a graph (V, E) is a map $c : V \rightarrow \{1, 2, 3, 4\}$ such that $(x, y) \in E$ implies $c(x) \neq c(y)$.

Theorem (4CT)

An finite planar graph has a 4-coloring.

Theorem

A planar graph $G = (V, E)$ with countably many vertices $V = \{v_1, v_2, \dots\}$ has a 4-coloring.

Proof $G \rightsquigarrow$ set of formulas S s.t. S is sat. iff G is 4-col.

An application: Graph Coloring

Definition

A **4-coloring** of a graph (V, E) is a map $c : V \rightarrow \{1, 2, 3, 4\}$ such that $(x, y) \in E$ implies $c(x) \neq c(y)$.

Theorem (4CT)

An finite planar graph has a 4-coloring.

Theorem

A planar graph $G = (V, E)$ with countably many vertices $V = \{v_1, v_2, \dots\}$ has a 4-coloring.

Proof $G \rightsquigarrow$ set of formulas S s.t. S is sat. iff G is 4-col.

G is planar

\Rightarrow every finite subgraph of G is planar and 4-col. (by 4CT)

An application: Graph Coloring

Definition

A **4-coloring** of a graph (V, E) is a map $c : V \rightarrow \{1, 2, 3, 4\}$ such that $(x, y) \in E$ implies $c(x) \neq c(y)$.

Theorem (4CT)

An finite planar graph has a 4-coloring.

Theorem

A planar graph $G = (V, E)$ with countably many vertices $V = \{v_1, v_2, \dots\}$ has a 4-coloring.

Proof $G \rightsquigarrow$ set of formulas S s.t. S is sat. iff G is 4-col.

G is planar

\Rightarrow every finite subgraph of G is planar and 4-col. (by 4CT)

\Rightarrow every finite subset of S is sat.

An application: Graph Coloring

Definition

A **4-coloring** of a graph (V, E) is a map $c : V \rightarrow \{1, 2, 3, 4\}$ such that $(x, y) \in E$ implies $c(x) \neq c(y)$.

Theorem (4CT)

An finite planar graph has a 4-coloring.

Theorem

A planar graph $G = (V, E)$ with countably many vertices $V = \{v_1, v_2, \dots\}$ has a 4-coloring.

Proof $G \rightsquigarrow$ set of formulas S s.t. S is sat. iff G is 4-col.

G is planar

\Rightarrow every finite subgraph of G is planar and 4-col. (by 4CT)

\Rightarrow every finite subset of S is sat.

$\Rightarrow S$ is sat. (by Compactness)

An application: Graph Coloring

Definition

A **4-coloring** of a graph (V, E) is a map $c : V \rightarrow \{1, 2, 3, 4\}$ such that $(x, y) \in E$ implies $c(x) \neq c(y)$.

Theorem (4CT)

An finite planar graph has a 4-coloring.

Theorem

A planar graph $G = (V, E)$ with countably many vertices $V = \{v_1, v_2, \dots\}$ has a 4-coloring.

Proof $G \rightsquigarrow$ set of formulas S s.t. S is sat. iff G is 4-col.

G is planar

\Rightarrow every finite subgraph of G is planar and 4-col. (by 4CT)

\Rightarrow every finite subset of S is sat.

$\Rightarrow S$ is sat. (by Compactness)

$\Rightarrow G$ is 4-col.

Proof details

$$G \rightsquigarrow S:$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid i \in \mathbb{N},$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid i \in \mathbb{N}, c, d \in \{1, \dots, 4\},$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid i \in \mathbb{N}, c, d \in \{1, \dots, 4\}, c \neq d\} \cup$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid i \in \mathbb{N}, c, d \in \{1, \dots, 4\}, c \neq d\} \cup \\ \{\neg(A_i^c \wedge A_j^c) \mid$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid i \in \mathbb{N}, c, d \in \{1, \dots, 4\}, c \neq d\} \cup \\ \{\neg(A_i^c \wedge A_j^c) \mid (v_i, v_j) \in E,$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid i \in \mathbb{N}, c, d \in \{1, \dots, 4\}, c \neq d\} \cup \\ \{\neg(A_i^c \wedge A_j^c) \mid (v_i, v_j) \in E, c \in \{1, \dots, 4\}\}$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid i \in \mathbb{N}, c, d \in \{1, \dots, 4\}, c \neq d\} \cup \\ \{\neg(A_i^c \wedge A_j^c) \mid (v_i, v_j) \in E, c \in \{1, \dots, 4\}\}$$

Subgraph corresponding to some $T \subseteq S$:

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid i \in \mathbb{N}, c, d \in \{1, \dots, 4\}, c \neq d\} \cup \\ \{\neg(A_i^c \wedge A_j^c) \mid (v_i, v_j) \in E, c \in \{1, \dots, 4\}\}$$

Subgraph corresponding to some $T \subseteq S$:

$$V_T := \{v_i \mid A_i^c \text{ occurs in } T \text{ (for some } c)\}$$

Proof details

$G \rightsquigarrow S$:

For simplicity:

atoms are of the form A_i^c where $c \in \{1, \dots, 4\}$ and $i \in \mathbb{N}$

$$S := \{A_i^1 \vee A_i^2 \vee A_i^3 \vee A_i^4 \mid i \in \mathbb{N}\} \cup \\ \{A_i^c \rightarrow \neg A_i^d \mid i \in \mathbb{N}, c, d \in \{1, \dots, 4\}, c \neq d\} \cup \\ \{\neg(A_i^c \wedge A_j^c) \mid (v_i, v_j) \in E, c \in \{1, \dots, 4\}\}$$

Subgraph corresponding to some $T \subseteq S$:

$$V_T := \{v_i \mid A_i^c \text{ occurs in } T \text{ (for some } c)\}$$

$$E_T := \{(v_i, v_j) \mid \neg(A_i^c \wedge A_j^c) \in T \text{ (for some } c)\}$$

Proof of Compactness

Theorem

*A set S of formulas is satisfiable
iff every finite subset of S is satisfiable.*

Proof of Compactness

Theorem

*A set S of formulas is satisfiable
iff every finite subset of S is satisfiable.*

Proof

Proof of Compactness

Theorem

*A set S of formulas is satisfiable
iff every finite subset of S is satisfiable.*

Proof

\Rightarrow : If S is satisfiable then every finite subset of S is satisfiable.

Proof of Compactness

Theorem

*A set S of formulas is satisfiable
iff every finite subset of S is satisfiable.*

Proof

\Rightarrow : If S is satisfiable then every finite subset of S is satisfiable.

Trivial.

Proof of Compactness

Theorem

*A set S of formulas is satisfiable
iff every finite subset of S is satisfiable.*

Proof

\Rightarrow : If S is satisfiable then every finite subset of S is satisfiable.

Trivial.

\Leftarrow : If every finite subset of S is satisfiable then S is satisfiable.

Proof of Compactness

Theorem

*A set S of formulas is satisfiable
iff every finite subset of S is satisfiable.*

Proof

\Rightarrow : If S is satisfiable then every finite subset of S is satisfiable.

Trivial.

\Leftarrow : If every finite subset of S is satisfiable then S is satisfiable.

We prove that S has a model.

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$:

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$: because all finite $T \subseteq S$ are satisfiable.

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$: because all finite $T \subseteq S$ are satisfiable.

Case $n + 1$:

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$: because all finite $T \subseteq S$ are satisfiable.

Case $n + 1$: We need to show that a suitable b exists.

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$: because all finite $T \subseteq S$ are satisfiable.

Case $n + 1$: We need to show that a suitable b exists.

Proof by contradiction. Assume there is no suitable b .

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$: because all finite $T \subseteq S$ are satisfiable.

Case $n + 1$: We need to show that a suitable b exists.

Proof by contradiction. Assume there is no suitable b .

Then there is a finite $T_0 \subseteq S$ that has no $b_1, \dots, b_n, 0$ model (0)

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$: because all finite $T \subseteq S$ are satisfiable.

Case $n + 1$: We need to show that a suitable b exists.

Proof by contradiction. Assume there is no suitable b .

Then there is a finite $T_0 \subseteq S$ that has no $b_1, \dots, b_n, 0$ model (0)
and there is a finite $T_1 \subseteq S$ that has no $b_1, \dots, b_n, 1$ model (1).

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$: because all finite $T \subseteq S$ are satisfiable.

Case $n + 1$: We need to show that a suitable b exists.

Proof by contradiction. Assume there is no suitable b .

Then there is a finite $T_0 \subseteq S$ that has no $b_1, \dots, b_n, 0$ model (0)
and there is a finite $T_1 \subseteq S$ that has no $b_1, \dots, b_n, 1$ model (1).

Therefore $T_0 \cup T_1$ has no b_1, \dots, b_n model \mathcal{A} :

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$: because all finite $T \subseteq S$ are satisfiable.

Case $n + 1$: We need to show that a suitable b exists.

Proof by contradiction. Assume there is no suitable b .

Then there is a finite $T_0 \subseteq S$ that has no $b_1, \dots, b_n, 0$ model (0)

and there is a finite $T_1 \subseteq S$ that has no $b_1, \dots, b_n, 1$ model (1).

Therefore $T_0 \cup T_1$ has no b_1, \dots, b_n model \mathcal{A} :

$\mathcal{A}(A_{n+1}) = 0$ contradicts (0), $\mathcal{A}(A_{n+1}) = 1$ contradicts (1).

Proof of Compactness

Terminology: \mathcal{A} is a b_1, \dots, b_n model of T
(where $b_1, \dots, b_n \in \{0, 1\}^*$ and T is a set of formulas)
if $\mathcal{A}(A_i) = b_i$ (for $i = 1, \dots, n$) and $\mathcal{A} \models T$.

Define an infinite sequence b_1, b_2, \dots recursively as follows:

$b_{n+1} =$ some $b \in \{0, 1\}$ s.t.
all finite $T \subseteq S$ have a b_1, \dots, b_n, b model.

Claim 1: For all n , all finite $T \subseteq S$ have a b_1, \dots, b_n model.

Proof by induction on n .

Case $n = 0$: because all finite $T \subseteq S$ are satisfiable.

Case $n + 1$: We need to show that a suitable b exists.

Proof by contradiction. Assume there is no suitable b .

Then there is a finite $T_0 \subseteq S$ that has no $b_1, \dots, b_n, 0$ model (0)

and there is a finite $T_1 \subseteq S$ that has no $b_1, \dots, b_n, 1$ model (1).

Therefore $T_0 \cup T_1$ has no b_1, \dots, b_n model \mathcal{A} :

$\mathcal{A}(A_{n+1}) = 0$ contradicts (0), $\mathcal{A}(A_{n+1}) = 1$ contradicts (1).

But by IH: $T_0 \cup T_1$ has a b_1, \dots, b_n model — Contradiction!

Proof of Compactness

Define $\mathcal{B}(A_i) = b_i$ for all i .

Proof of Compactness

Define $\mathcal{B}(A_i) = b_i$ for all i .

Claim 2: $\mathcal{B} \models S$

Proof of Compactness

Define $\mathcal{B}(A_i) = b_i$ for all i .

Claim 2: $\mathcal{B} \models S$

We show $\mathcal{B} \models F$ for all $F \in S$.

Proof of Compactness

Define $\mathcal{B}(A_i) = b_i$ for all i .

Claim 2: $\mathcal{B} \models S$

We show $\mathcal{B} \models F$ for all $F \in S$.

Let m be the maximal index of all atoms in F .

Proof of Compactness

Define $\mathcal{B}(A_i) = b_i$ for all i .

Claim 2: $\mathcal{B} \models S$

We show $\mathcal{B} \models F$ for all $F \in S$.

Let m be the maximal index of all atoms in F .

By Claim 1, $\{F\}$ has a b_1, \dots, b_m model \mathcal{A} .

Proof of Compactness

Define $\mathcal{B}(A_i) = b_i$ for all i .

Claim 2: $\mathcal{B} \models S$

We show $\mathcal{B} \models F$ for all $F \in S$.

Let m be the maximal index of all atoms in F .

By Claim 1, $\{F\}$ has a b_1, \dots, b_m model \mathcal{A} .

Hence $\mathcal{B} \models F$ because \mathcal{A} and \mathcal{B} agree on all atoms in F .

Corollary

Corollary

If $S \models F$ then there is a finite subset $M \subseteq S$ such that $M \models F$.

Propositional Logic
DPLL: Davis-Putnam-
Logemann-Loveland

Davis–Putnam–Logemann–Loveland

DPLL algorithm:

- ▶ combines search and deduction to decide satisfiability
- ▶ underlies most modern SAT solvers
- ▶ is over 50 years old



Davis–Putnam–Logemann–Loveland

DPLL algorithm:

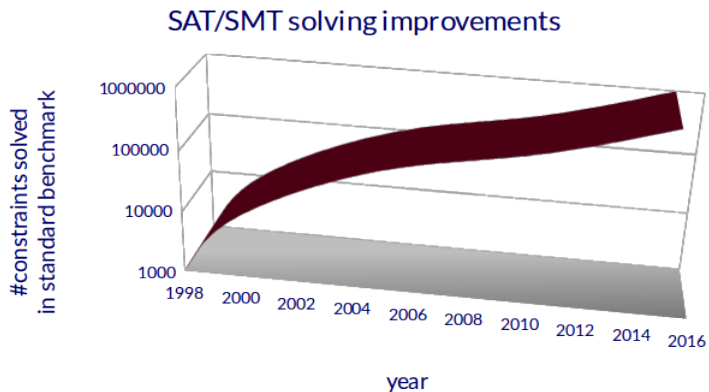
- ▶ combines search and deduction to decide satisfiability
- ▶ underlies most modern SAT solvers
- ▶ is over 50 years old



DPLL-based SAT solvers \geq 1990:

- ▶ clause learning
- ▶ non-chronological backtracking
- ▶ branching heuristics
- ▶ lazy evaluation

Performance increase of SAT solvers



Clause representation of CNF formulas

Clause representation of CNF formulas

$$\text{CNF: } (L_{1,1} \vee \dots \vee L_{1,n_1}) \wedge \dots \wedge (L_{k,1} \vee \dots \vee L_{1,n_k})$$

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \dots \vee L_{1,n_1}) \wedge \dots \wedge (L_{k,1} \vee \dots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\underbrace{\{\{L_{1,1}, \dots, L_{1,n_1}\}, \dots, \{L_{k,1}, \dots, L_{1,n_k}\}\}}_{\text{clause}}$$

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \dots \vee L_{1,n_1}) \wedge \dots \wedge (L_{k,1} \vee \dots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\underbrace{\{\{L_{1,1}, \dots, L_{1,n_1}\}, \dots, \{L_{k,1}, \dots, L_{1,n_k}\}\}}_{\text{clause}}$$

Clause = set of literals (disjunction).

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \dots \vee L_{1,n_1}) \wedge \dots \wedge (L_{k,1} \vee \dots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\underbrace{\{\{L_{1,1}, \dots, L_{1,n_1}\}, \dots, \{L_{k,1}, \dots, L_{1,n_k}\}\}}_{\text{clause}}$$

Clause = set of literals (disjunction).

Formula in CNF = set of clauses

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \dots \vee L_{1,n_1}) \wedge \dots \wedge (L_{k,1} \vee \dots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\underbrace{\{L_{1,1}, \dots, L_{1,n_1}\}}_{\text{clause}}, \dots, \{L_{k,1}, \dots, L_{1,n_k}\}$$

Clause = set of literals (disjunction).

Formula in CNF = set of clauses

Degenerate cases:

The empty clause stands for \perp .

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \dots \vee L_{1,n_1}) \wedge \dots \wedge (L_{k,1} \vee \dots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\underbrace{\{\{L_{1,1}, \dots, L_{1,n_1}\}, \dots, \{L_{k,1}, \dots, L_{1,n_k}\}\}}_{\text{clause}}$$

Clause = set of literals (disjunction).

Formula in CNF = set of clauses

Degenerate cases:

The empty clause stands for \perp .

The empty set of clauses stands for \top .

The joy of sets

We get “for free”:

The joy of sets

We get “for free”:

▶ **Commutativity:**

$A \vee B \equiv B \vee A$, both represented by $\{A, B\}$

The joy of sets

We get “for free”:

▶ **Commutativity:**

$A \vee B \equiv B \vee A$, both represented by $\{A, B\}$

▶ **Associativity:**

$(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$

The joy of sets

We get “for free”:

▶ **Commutativity:**

$A \vee B \equiv B \vee A$, both represented by $\{A, B\}$

▶ **Associativity:**

$(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$

▶ **Idempotence:**

$(A \vee A) \equiv A$, both represented by $\{A\}$

The joy of sets

We get “for free”:

▶ **Commutativity:**

$A \vee B \equiv B \vee A$, both represented by $\{A, B\}$

▶ **Associativity:**

$(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$

▶ **Idempotence:**

$(A \vee A) \equiv A$, both represented by $\{A\}$

Sets are a convenient representation of conjunctions and disjunctions that build in associativity, commutativity and idempotence

CNF-SAT: Input: Set of clauses F

The joy of sets

We get “for free”:

▶ **Commutativity:**

$A \vee B \equiv B \vee A$, both represented by $\{A, B\}$

▶ **Associativity:**

$(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$

▶ **Idempotence:**

$(A \vee A) \equiv A$, both represented by $\{A\}$

Sets are a convenient representation of conjunctions and disjunctions that build in associativity, commutativity and idempotence

CNF-SAT: Input: Set of clauses F

Question: Is F unsatisfiable?

The joy of sets

We get “for free”:

▶ **Commutativity:**

$A \vee B \equiv B \vee A$, both represented by $\{A, B\}$

▶ **Associativity:**

$(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$

▶ **Idempotence:**

$(A \vee A) \equiv A$, both represented by $\{A\}$

Sets are a convenient representation of conjunctions and disjunctions that build in associativity, commutativity and idempotence

CNF-SAT: Input: Set of clauses F

Question: Is F unsatisfiable?

DPLL — First step: partial evaluation

Simplest algorithm: Construct the truth table.

Best-case runtime is $\Theta(m \cdot 2^n)$ for a formula of length m over n variables.

Improvement: **partial evaluation** using Boole-Shannon expansion

Lemma (Boole-Shannon Expansion)

Let $F[\perp/A]$ and $F[\top/A]$ be the result of substituting \perp and \top for A in F , respectively. Then:

$$F \equiv (A \wedge F[\top/A]) \vee (\neg A \wedge F[\perp/A]).$$

Proof By structural induction on F (exercise).

Corollary

F is satisfiable iff $F[\perp/A]$ or $F[\top/A]$ are unsatisfiable.

DPLL — First step: partial evaluation

$F[\perp/A]$ and $F[\top/A]$ easy to compute in clause normal form:

$F[\top/A] \equiv$ take F , remove all clauses with A , remove all $\neg A$.

DPLL — First step: partial evaluation

$F[\perp/A]$ and $F[\top/A]$ easy to compute in clause normal form:

$F[\top/A] \equiv$ take F , remove all clauses with A , remove all $\neg A$.

$F[\perp/A] \equiv$ take F , remove all clauses with $\neg A$, remove all A .

Partial evaluation algorithm:

Given formula F , total order on the variables \prec :

If $\{\} \in F$ return **unsatisfiable**.

If $F = \emptyset$ return **satisfiable**.

Otherwise:

Fix the first variable A in F according to \prec .

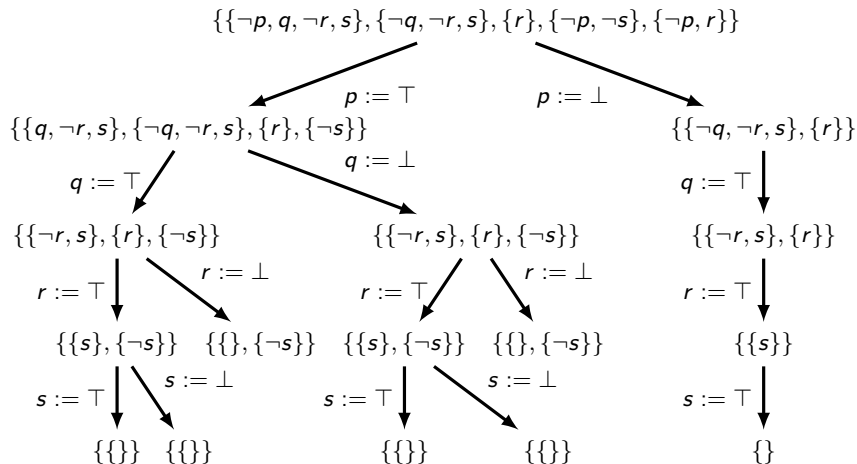
Recursively check if $F[\perp/A]$ is satisfiable;

if yes, return **satisfiable**.

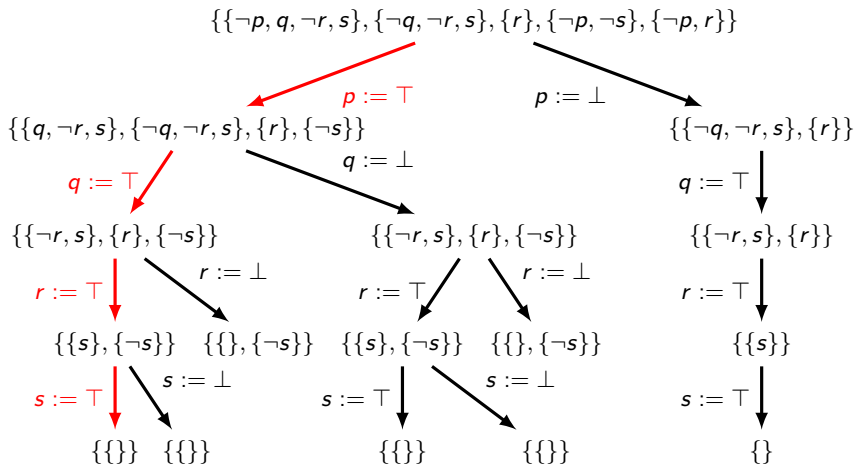
Recursively check if $F[\top/A]$ is satisfiable;

if yes, return **satisfiable**, otherwise **unsatisfiable**.

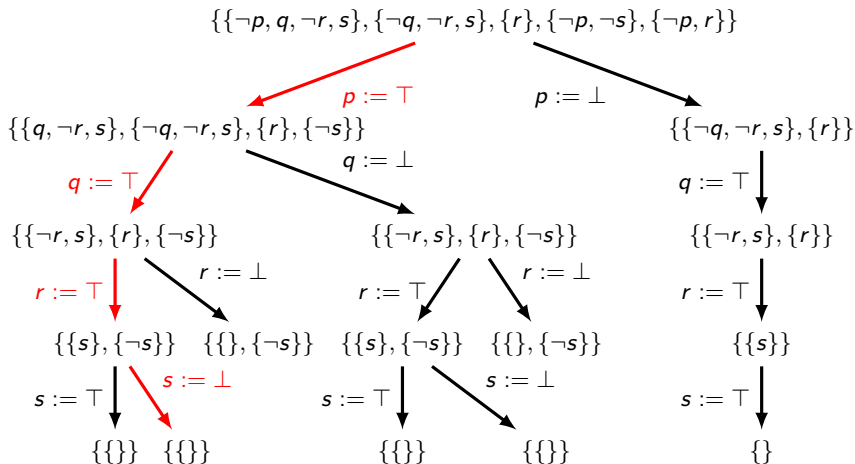
DPLL: Davis-Putnam-Logemann-Loveland



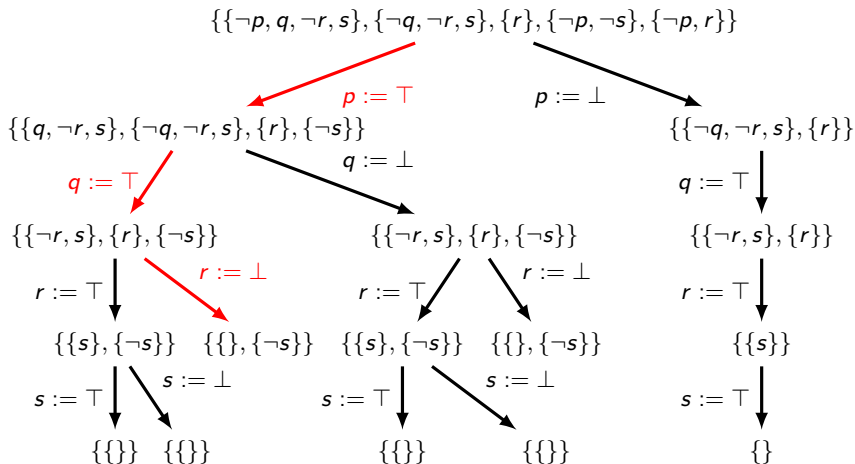
DPLL: Davis-Putnam-Logemann-Loveland



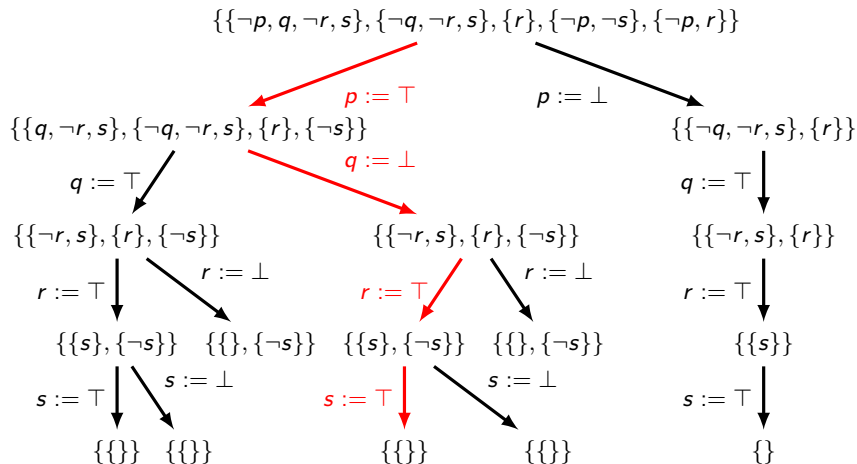
DPLL: Davis-Putnam-Logemann-Loveland



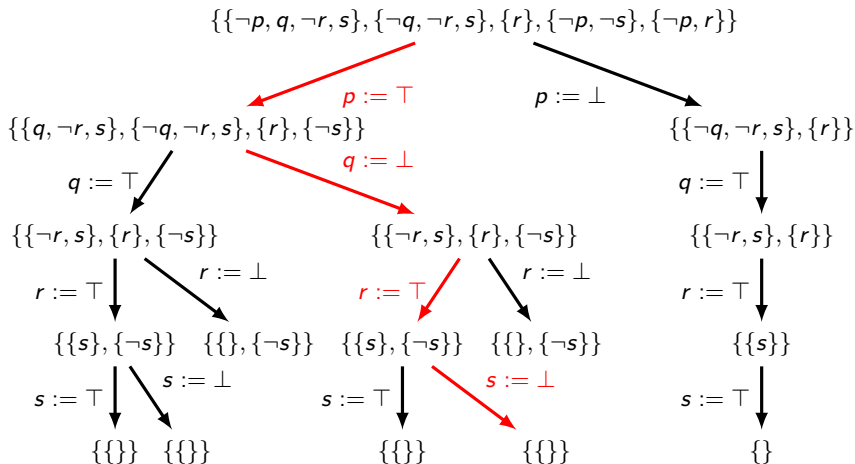
DPLL: Davis-Putnam-Logemann-Loveland



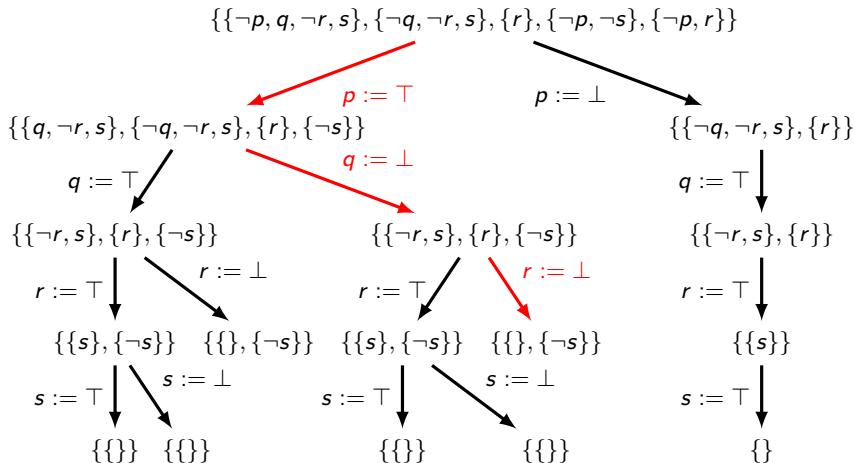
DPLL: Davis-Putnam-Logemann-Loveland



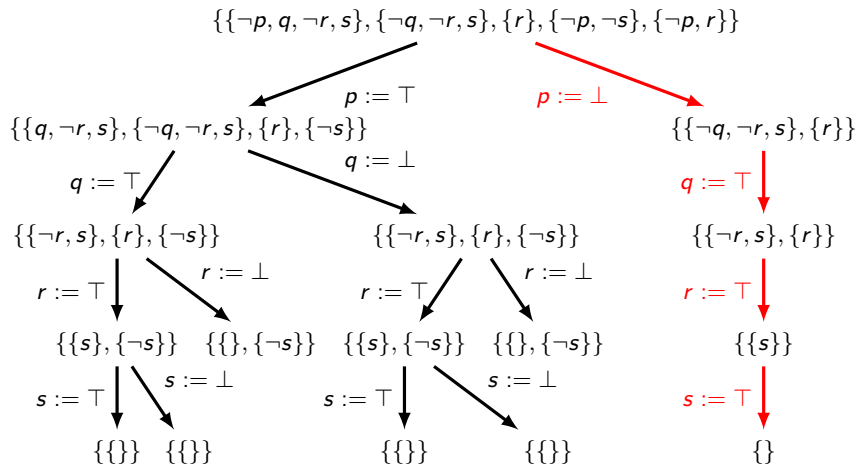
DPLL: Davis-Putnam-Logemann-Loveland



DPLL: Davis-Putnam-Logemann-Loveland



DPLL: Davis-Putnam-Logemann-Loveland



DPLL: Davis-Putnam-Logemann-Loveland

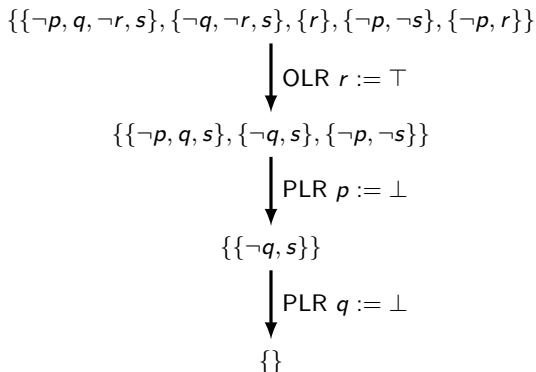
Instead of fixing an order on variables, choose the next variable **dynamically**.

- ▶ **OLR**: one-literal rule If $\{L\} \in F$ ($\{L\}$ is called **unit clause**), then every satisfying assignment sets L to true. So it suffices to check satisfiability of $F[\top/L]$.
- ▶ **PLR**: pure-literal rule
If L appears in F and \bar{L} does not, then it also suffices to check satisfiability of $F[\top/L]$ (**Why?**).

DPLL algorithm: Partial evaluation that gives priority to a variable satisfying **OLR**, then to a variable satisfying **PLR**, and otherwise picks the first unpicked variable of \prec .

Applying **OLR** can generate further unit clauses (**unit propagation**). Same for **PLR**, but DPLL often implemented with only **OLR** for efficiency.

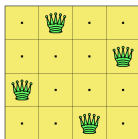
DPLL: Davis-Putnam-Logemann-Loveland



In this example PLR and OLR allow us to avoid all case splits.

Example: 4 queens

Problem: place 4 non-attacking queens on a 4x4 chess board



Variable p_{ij} models: there is a queen in square (i, j)

- ▶ ≥ 1 in each row: $\bigwedge_{i=1}^4 \bigvee_{j=1}^4 p_{ij}$
- ▶ ≤ 1 in each row: $\bigwedge_{i=1}^4 \bigwedge_{j \neq j'=1}^4 \neg p_{ij} \vee \neg p_{ij'}$
- ▶ ≤ 1 in each column: $\bigwedge_{j=1}^4 \bigwedge_{i \neq i'=1}^4 \neg p_{ij} \vee \neg p_{i'j}$
- ▶ ≤ 1 on each diagonal: $\bigwedge_{i,j=1}^4 \bigvee_k \neg p_{i-k,i+k} \vee \neg p_{i+k,j+k}$

Total number of clauses: $4 + 24 + 24 + 28 = 80$

DPLL: 4 queens

Running the DPLL algorithm:

- ▶ Start with $p_{11} \mapsto 1$
delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, delete $\neg p_{11}$: 9 new unit clauses
unit propagation: deletes 65 clauses!

DPLL: 4 queens

Running the DPLL algorithm:

- ▶ Start with $p_{11} \mapsto 1$
delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, delete $\neg p_{11}$: 9 new unit clauses
unit propagation: deletes 65 clauses!
- ▶ Set $p_{23} \mapsto 1$
4 new unit clauses: $\{\neg p_{24}\}, \{\neg p_{43}\}, \{\neg p_{32}\}, \{\neg p_{34}\}$
unit propagation of $\{\neg p_{34}\}$: UNSAT

DPLL: 4 queens

Running the DPLL algorithm:

- ▶ Start with $p_{11} \mapsto 1$
delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, delete $\neg p_{11}$: 9 new unit clauses
unit propagation: deletes 65 clauses!
- ▶ Set $p_{23} \mapsto 1$
4 new unit clauses: $\{\neg p_{24}\}, \{\neg p_{43}\}, \{\neg p_{32}\}, \{\neg p_{34}\}$
unit propagation of $\{\neg p_{34}\}$: UNSAT
fixing only two literals collapsed from 80 clauses to 1
ruled out 2^{14} of 2^{16} possible assignments!
- ▶ Backtrack: $p_{11} \mapsto 0, p_{12} \mapsto 1$
delete $\{\neg p_{12}\}$: 9 new unit clauses
unit propagation: leaves only 1 clause $\{p_{43}\}$!

DPLL: 4 queens

Running the DPLL algorithm:

- ▶ Start with $p_{11} \mapsto 1$
delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, delete $\neg p_{11}$: 9 new unit clauses
unit propagation: deletes 65 clauses!
- ▶ Set $p_{23} \mapsto 1$
4 new unit clauses: $\{\neg p_{24}\}, \{\neg p_{43}\}, \{\neg p_{32}\}, \{\neg p_{34}\}$
unit propagation of $\{\neg p_{34}\}$: UNSAT
fixing only two literals collapsed from 80 clauses to 1
ruled out 2^{14} of 2^{16} possible assignments!
- ▶ Backtrack: $p_{11} \mapsto 0, p_{12} \mapsto 1$
delete $\{\neg p_{12}\}$: 9 new unit clauses
unit propagation: leaves only 1 clause $\{p_{43}\}$!
- ▶ Answer: $p_{12}, p_{24}, p_{31}, p_{43} \mapsto 1$

DPLL: Evaluation

Oriented towards satisfiability:

- ▶ $2^{O(n)}$ time for satisfiable formulas, but $2^{\Theta(n)}$ for unsatisfiable ones.
- ▶ DPLL computes a satisfying assignment, if there is one.
- ▶ The satisfying assignment is a **certificate** of satisfiability.
- ▶ Satisfiable formulas have short certificates: satisfying assignment never larger than the formula.

Coming next: **resolution**, a procedure oriented towards unsatisfiability.

- ▶ $2^{O(n)}$ time for unsatisfiable formulas, but $2^{\Theta(n)}$ for satisfiable ones.
- ▶ Resolution computes a certificate of unsatisfiability.
- ▶ However, the certificate is **exponentially longer** than the formula in the worst case.
- ▶ Polynomial certificates for satisfiability implies $NP = coNP$.

Propositional Logic Resolution

Resolution — The idea

Input: Set of clauses F

Resolution — The idea

Input: Set of clauses F

Question: Is F unsatisfiable?

Resolution — The idea

Input: Set of clauses F

Question: Is F unsatisfiable?

Algorithm:

Keep on “resolving” two clauses from F and adding the result to F

Resolution — The idea

Input: Set of clauses F

Question: Is F unsatisfiable?

Algorithm:

Keep on “resolving” two clauses from F and adding the result to F until the empty clause is found

Resolution — The idea

Input: Set of clauses F

Question: Is F unsatisfiable?

Algorithm:

Keep on “resolving” two clauses from F and adding the result to F until the empty clause is found

Correctness:

If the empty clause is found, the initial F is unsatisfiable

Resolution — The idea

Input: Set of clauses F

Question: Is F unsatisfiable?

Algorithm:

Keep on “resolving” two clauses from F and adding the result to F until the empty clause is found

Correctness:

If the empty clause is found, the initial F is unsatisfiable

Completeness:

If the initial F is unsatisfiable, the empty clause can be found.

Resolution — The idea

Input: Set of clauses F

Question: Is F unsatisfiable?

Algorithm:

Keep on “resolving” two clauses from F and adding the result to F until the empty clause is found

Correctness:

If the empty clause is found, the initial F is unsatisfiable

Completeness:

If the initial F is unsatisfiable, the empty clause can be found.

Correctness/Completeness of **syntactic** procedure (**resolution**)
w.r.t. **semantic** property (**unsatisfiability**)

Resolvent

Resolvent

Definition

Let L be a literal. Then \bar{L} is defined as follows:

$$\bar{L} = \begin{cases} \neg A_i & \text{if } L = A_i \\ A_i & \text{if } L = \neg A_i \end{cases}$$

Resolvent

Definition

Let L be a literal. Then \bar{L} is defined as follows:

$$\bar{L} = \begin{cases} \neg A_i & \text{if } L = A_i \\ A_i & \text{if } L = \neg A_i \end{cases}$$

Definition

Let C_1, C_2 be clauses and let L be a literal such that $L \in C_1$ and $\bar{L} \in C_2$. Then the clause

$$(C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$$

is a **resolvent** of C_1 and C_2 .

Resolvent

Definition

Let L be a literal. Then \bar{L} is defined as follows:

$$\bar{L} = \begin{cases} \neg A_i & \text{if } L = A_i \\ A_i & \text{if } L = \neg A_i \end{cases}$$

Definition

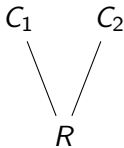
Let C_1, C_2 be clauses and let L be a literal such that $L \in C_1$ and $\bar{L} \in C_2$. Then the clause

$$(C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$$

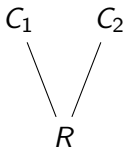
is a **resolvent** of C_1 and C_2 .

The process of deriving the resolvent is called a **resolution step**.

Graphical representation of resolvent:

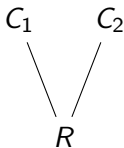


Graphical representation of resolvent:



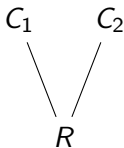
If $C_1 = \{L\}$ and $C_2 = \{\bar{L}\}$ then the empty clause is a resolvent of C_1 and C_2 .

Graphical representation of resolvent:



If $C_1 = \{L\}$ and $C_2 = \{\bar{L}\}$ then the empty clause is a resolvent of C_1 and C_2 . The special symbol \square denotes the empty clause.

Graphical representation of resolvent:



If $C_1 = \{L\}$ and $C_2 = \{\bar{L}\}$ then the empty clause is a resolvent of C_1 and C_2 . The special symbol \square denotes the empty clause.

Recall: \square represents \perp .

Resolution proof

Definition

A **resolution proof** of a clause C from a set of clauses F is a sequence of clauses C_0, \dots, C_n such that

Resolution proof

Definition

A **resolution proof** of a clause C from a set of clauses F is a sequence of clauses C_0, \dots, C_n such that

- ▶ $C_i \in F$ or C_i is a resolvent of two clauses C_a and C_b , $a, b < i$,

Resolution proof

Definition

A **resolution proof** of a clause C from a set of clauses F is a sequence of clauses C_0, \dots, C_n such that

- ▶ $C_i \in F$ or C_i is a resolvent of two clauses C_a and C_b , $a, b < i$,
- ▶ $C_n = C$

Resolution proof

Definition

A **resolution proof** of a clause C from a set of clauses F is a sequence of clauses C_0, \dots, C_n such that

- ▶ $C_i \in F$ or C_i is a resolvent of two clauses C_a and C_b , $a, b < i$,
- ▶ $C_n = C$

Then we can write $F \vdash_{Res} C$.

Resolution proof

Definition

A **resolution proof** of a clause C from a set of clauses F is a sequence of clauses C_0, \dots, C_n such that

- ▶ $C_i \in F$ or C_i is a resolvent of two clauses C_a and C_b , $a, b < i$,
- ▶ $C_n = C$

Then we can write $F \vdash_{Res} C$.

Note: F can be finite or infinite

Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

Example

$\{P, Q\}$

$\{P, \neg Q\}$

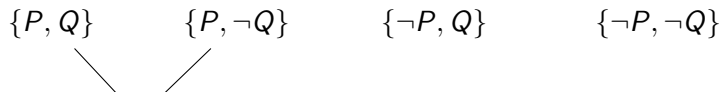
$\{\neg P, Q\}$

$\{\neg P, \neg Q\}$

Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

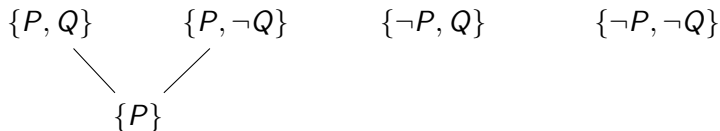
Example



Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

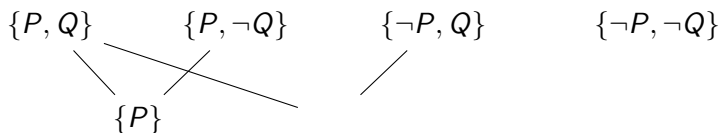
Example



Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

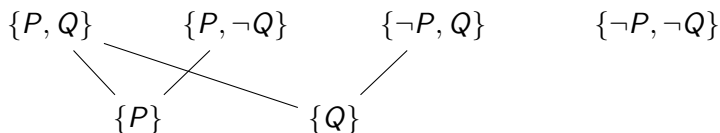
Example



Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

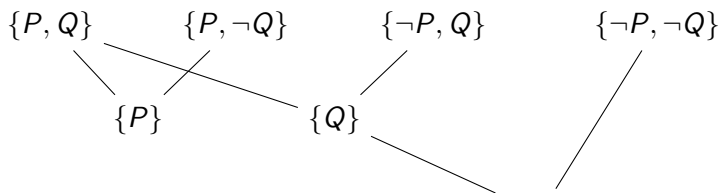
Example



Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

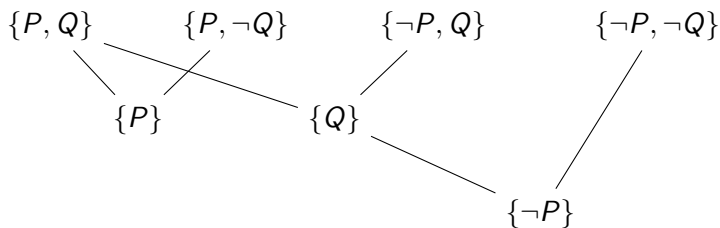
Example



Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

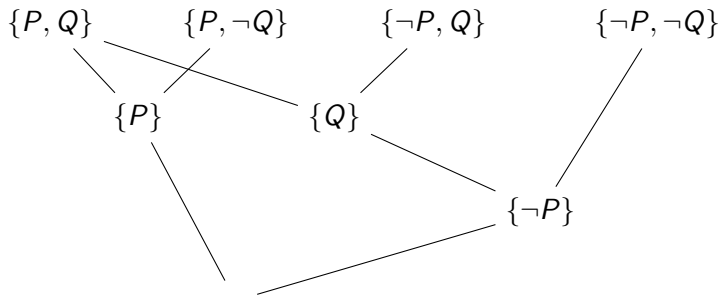
Example



Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

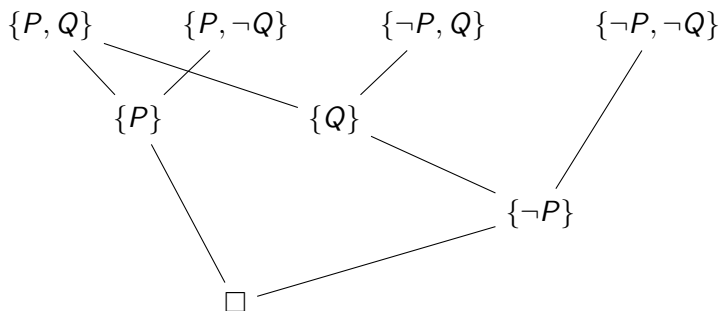
Example



Resolution proof as DAG

A resolution proof can be shown as a DAG with the clauses in F as the leaves and C as the root:

Example



A linear resolution proof

0: $\{P, Q\}$

1: $\{P, \neg Q\}$

2: $\{\neg P, Q\}$

3: $\{\neg P, \neg Q\}$

4: $\{P\}$ (0, 1)

5: $\{Q\}$ (0, 2)

6: $\{\neg P\}$ (3, 5)

7: \square (4, 6)

Correctness of resolution

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 .

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$),

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$),

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Proof Assume there is a resolution proof $C_0, \dots, C_n = C$.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Proof Assume there is a resolution proof $C_0, \dots, C_n = C$.

By induction on i we show $F \models C_i$.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Proof Assume there is a resolution proof $C_0, \dots, C_n = C$.

By induction on i we show $F \models C_i$. IH: $F \models C_j$ for all $j < i$.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Proof Assume there is a resolution proof $C_0, \dots, C_n = C$.

By induction on i we show $F \models C_i$. IH: $F \models C_j$ for all $j < i$.

If $C_i \in F$ then $F \models C_i$ is trivial.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Proof Assume there is a resolution proof $C_0, \dots, C_n = C$.

By induction on i we show $F \models C_i$. IH: $F \models C_j$ for all $j < i$.

If $C_i \in F$ then $F \models C_i$ is trivial. If C_i is a resolvent of C_a and C_b , $a, b < i$,

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Proof Assume there is a resolution proof $C_0, \dots, C_n = C$.

By induction on i we show $F \models C_i$. IH: $F \models C_j$ for all $j < i$.

If $C_i \in F$ then $F \models C_i$ is trivial. If C_i is a resolvent of C_a and C_b , $a, b < i$, then $F \models C_a$ and $F \models C_b$ by IH

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Proof Assume there is a resolution proof $C_0, \dots, C_n = C$.

By induction on i we show $F \models C_i$. IH: $F \models C_j$ for all $j < i$.

If $C_i \in F$ then $F \models C_i$ is trivial. If C_i is a resolvent of C_a and C_b , $a, b < i$, then $F \models C_a$ and $F \models C_b$ by IH and $C_a, C_b \models C_i$ by the resolution lemma.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Proof Assume there is a resolution proof $C_0, \dots, C_n = C$.

By induction on i we show $F \models C_i$. IH: $F \models C_j$ for all $j < i$.

If $C_i \in F$ then $F \models C_i$ is trivial. If C_i is a resolvent of C_a and C_b , $a, b < i$, then $F \models C_a$ and $F \models C_b$ by IH and $C_a, C_b \models C_i$ by the resolution lemma. Thus $F \models C_i$.

Correctness of resolution

Lemma (Resolution Lemma)

Let R be a resolvent of two clauses C_1 and C_2 . Then $C_1, C_2 \models R$.

Proof By definition $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ (for some L).

Let $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. There are two cases.

If $\mathcal{A} \models L$ then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.

If $\mathcal{A} \not\models L$ then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Theorem (Correctness of resolution)

Let F be a set of clauses. If $F \vdash_{Res} C$ then $F \models C$.

Proof Assume there is a resolution proof $C_0, \dots, C_n = C$.

By induction on i we show $F \models C_i$. IH: $F \models C_j$ for all $j < i$.

If $C_i \in F$ then $F \models C_i$ is trivial. If C_i is a resolvent of C_a and C_b , $a, b < i$, then $F \models C_a$ and $F \models C_b$ by IH and $C_a, C_b \models C_i$ by the resolution lemma. Thus $F \models C_i$.

Corollary

Let F be a set of clauses. If $F \vdash_{Res} \square$ then F is unsatisfiable.

Completeness of resolution

Theorem

Let F be a finite set of clauses. If F is unsatisfiable then $F \vdash_{\text{Res}} \square$.

Completeness of resolution

Theorem

Let F be a finite set of clauses. If F is unsatisfiable then $F \vdash_{Res} \square$.

Theorem (Completeness of resolution)

Let F be a set of clauses. If F is unsatisfiable then $F \vdash_{Res} \square$.

Completeness of resolution

Theorem

Let F be a finite set of clauses. If F is unsatisfiable then $F \vdash_{Res} \square$.

Theorem (Completeness of resolution)

Let F be a set of clauses. If F is unsatisfiable then $F \vdash_{Res} \square$.

Proof If F is infinite, there must be a finite unsatisfiable subset of F (by the Compactness Theorem);

Completeness of resolution

Theorem

Let F be a finite set of clauses. If F is unsatisfiable then $F \vdash_{Res} \square$.

Theorem (Completeness of resolution)

Let F be a set of clauses. If F is unsatisfiable then $F \vdash_{Res} \square$.

Proof If F is infinite, there must be a finite unsatisfiable subset of F (by the Compactness Theorem); in that case let F be that finite subset and apply the previous theorem.

Completeness of resolution

Theorem

Let F be a finite set of clauses. If F is unsatisfiable then $F \vdash_{Res} \square$.

Theorem (Completeness of resolution)

Let F be a set of clauses. If F is unsatisfiable then $F \vdash_{Res} \square$.

Proof If F is infinite, there must be a finite unsatisfiable subset of F (by the Compactness Theorem); in that case let F be that finite subset and apply the previous theorem.

Corollary

A set of clauses F is unsatisfiable iff $F \vdash_{Res} \square$.

Completeness proof

Lemma (Boole-Shannon Expansion)

Let $F[\perp/A]$ and $F[\top/A]$ be the result of substituting \perp and \top for A in F , respectively. Then:

$$F \equiv (A \wedge F[\top/A]) \vee (\neg A \wedge F[\perp/A]).$$

Proof By structural induction on F (exercise).

Corollary

F is unsatisfiable iff $F[\perp/A]$ and $F[\top/A]$ are unsatisfiable.

Idea for completeness proof: If A is an atom of F , then $F[\perp/A]$ and $F[\top/A]$ have fewer atoms than F . Use Boole-Shannon to prove completeness by induction on the number of atoms of F : Given F unsatisfiable, construct inductively resolution proofs for $F[\perp/A]$ and $F[\top/A]$ and “combine” them into a resolution proof for F .

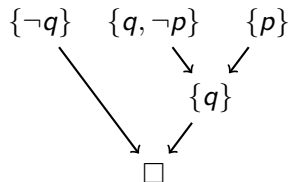
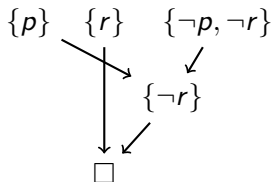
Inductive construction of resolution proofs

$$F = \{ \{ \neg q, s \}, \{ \neg p, q, s \}, \{ p \}, \{ r, \neg s \}, \{ \neg p, \neg r, \neg s \} \}$$

- Compute inductively proofs for $F[\top/s]$ and $F[\perp/s]$.

$$F[\top/s] \equiv \{ \{ p \}, \{ r \}, \{ \neg p, \neg r \} \}$$

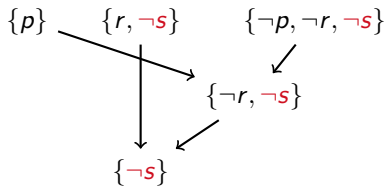
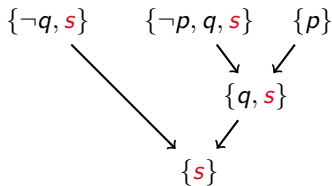
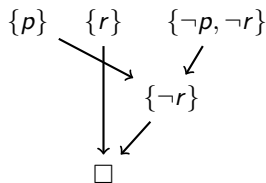
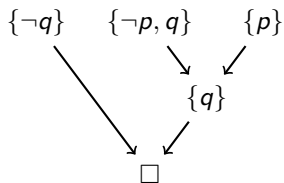
$$F[\perp/s] \equiv \{ \{ \neg q \}, \{ \neg p, q \}, \{ p \} \}$$



Inductive construction of resolution proofs

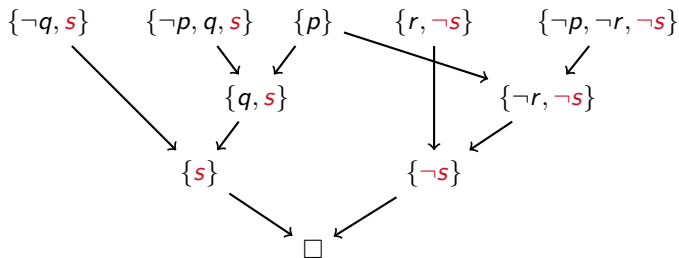
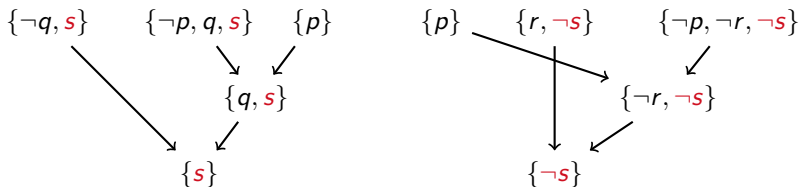
- ▶ Reintroduce s and $\neg s$.

$$F = \{ \{\neg q, s\}, \{\neg p, q, s\}, \{p\}, \{r, \neg s\}, \{\neg p, \neg r, \neg s\} \}$$



Inductive construction of resolution proofs

- Combine the graphs for $\{s\}$ and $\{\neg s\}$.



Completeness proof

Theorem

Let F be a finite set of clauses. If F is unsatisfiable then $F \vdash_{Res} \square$.

Completeness proof

Theorem

Let F be a finite set of clauses. *If F is unsatisfiable then $F \vdash_{Res} \square$.*

Proof The proof of $F \vdash_{Res} \square$ is by induction on the number n of distinct atoms in F .

Completeness proof

Theorem

Let F be a finite set of clauses. *If F is unsatisfiable then $F \vdash_{Res} \square$.*

Proof The proof of $F \vdash_{Res} \square$ is by induction on the number n of distinct atoms in F .

Basis: If $n = 0$ then $F = \{\}$

Completeness proof

Theorem

Let F be a finite set of clauses. *If F is unsatisfiable then $F \vdash_{Res} \square$.*

Proof The proof of $F \vdash_{Res} \square$ is by induction on the number n of distinct atoms in F .

Basis: If $n = 0$ then $F = \{\}$ (but F is unsat.)

Completeness proof

Theorem

Let F be a finite set of clauses. *If F is unsatisfiable then $F \vdash_{Res} \square$.*

Proof The proof of $F \vdash_{Res} \square$ is by induction on the number n of distinct atoms in F .

Basis: If $n = 0$ then $F = \{\}$ (but F is unsat.) or $F = \{\square\}$.

Completeness proof

Theorem

Let F be a finite set of clauses. *If F is unsatisfiable then $F \vdash_{Res} \square$.*

Proof The proof of $F \vdash_{Res} \square$ is by induction on the number n of distinct atoms in F .

Basis: If $n = 0$ then $F = \{\}$ (but F is unsat.) or $F = \{\square\}$.

Step:

IH: For every unsat. set of clauses F with n dist. atoms, $F \vdash_{Res} \square$.

Completeness proof

Theorem

Let F be a finite set of clauses. *If F is unsatisfiable then $F \vdash_{Res} \square$.*

Proof The proof of $F \vdash_{Res} \square$ is by induction on the number n of distinct atoms in F .

Basis: If $n = 0$ then $F = \{\}$ (but F is unsat.) or $F = \{\square\}$.

Step:

IH: For every unsat. set of clauses F with n dist. atoms, $F \vdash_{Res} \square$.

Let F contain $n + 1$ distinct atoms.

Completeness proof

Theorem

Let F be a finite set of clauses. *If F is unsatisfiable then $F \vdash_{Res} \square$.*

Proof The proof of $F \vdash_{Res} \square$ is by induction on the number n of distinct atoms in F .

Basis: If $n = 0$ then $F = \{\}$ (but F is unsat.) or $F = \{\square\}$.

Step:

IH: For every unsat. set of clauses F with n dist. atoms, $F \vdash_{Res} \square$.

Let F contain $n + 1$ distinct atoms. Pick some atom A in F .

Completeness proof

Theorem

Let F be a finite set of clauses. *If F is unsatisfiable then $F \vdash_{Res} \square$.*

Proof The proof of $F \vdash_{Res} \square$ is by induction on the number n of distinct atoms in F .

Basis: If $n = 0$ then $F = \{\}$ (but F is unsat.) or $F = \{\square\}$.

Step:

IH: For every unsat. set of clauses F with n dist. atoms, $F \vdash_{Res} \square$.

Let F contain $n + 1$ distinct atoms. Pick some atom A in F .

$F[\top/A] \equiv$ take F , remove all clauses with A , remove all $\neg A$.

Completeness proof

Theorem

Let F be a finite set of clauses. *If F is unsatisfiable then $F \vdash_{Res} \square$.*

Proof The proof of $F \vdash_{Res} \square$ is by induction on the number n of distinct atoms in F .

Basis: If $n = 0$ then $F = \{\}$ (but F is unsat.) or $F = \{\square\}$.

Step:

IH: For every unsat. set of clauses F with n dist. atoms, $F \vdash_{Res} \square$.

Let F contain $n + 1$ distinct atoms. Pick some atom A in F .

$F[\top/A] \equiv$ take F , remove all clauses with A , remove all $\neg A$.

$F[\perp/A] \equiv$ take F , remove all clauses with $\neg A$, remove all A .

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from.

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$
- ▶ or $C'_m = \square$ (and we are done).

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$
- ▶ or $C'_m = \square$ (and we are done).

Similarly we transform D_0, \dots, D_n into a proof D'_0, \dots, D'_n from F

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$
- ▶ or $C'_m = \square$ (and we are done).

Similarly we transform D_0, \dots, D_n into a proof D'_0, \dots, D'_n from F (by adding $\neg A$ back in).

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$
- ▶ or $C'_m = \square$ (and we are done).

Similarly we transform D_0, \dots, D_n into a proof D'_0, \dots, D'_n from F (by adding $\neg A$ back in).

Then $D'_n = \{\neg A\}$

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$
- ▶ or $C'_m = \square$ (and we are done).

Similarly we transform D_0, \dots, D_n into a proof D'_0, \dots, D'_n from F (by adding $\neg A$ back in).

Then $D'_n = \{\neg A\}$ or $D'_n = \square$

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$
- ▶ or $C'_m = \square$ (and we are done).

Similarly we transform D_0, \dots, D_n into a proof D'_0, \dots, D'_n from F (by adding $\neg A$ back in).

Then $D'_n = \{\neg A\}$ or $D'_n = \square$ (and we are done).

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$
- ▶ or $C'_m = \square$ (and we are done).

Similarly we transform D_0, \dots, D_n into a proof D'_0, \dots, D'_n from F (by adding $\neg A$ back in).

Then $D'_n = \{\neg A\}$ or $D'_n = \square$ (and we are done).

If $C'_m = \{A\}$ and $D'_n = \{\neg A\}$

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$
- ▶ or $C'_m = \square$ (and we are done).

Similarly we transform D_0, \dots, D_n into a proof D'_0, \dots, D'_n from F (by adding $\neg A$ back in).

Then $D'_n = \{\neg A\}$ or $D'_n = \square$ (and we are done).

If $C'_m = \{A\}$ and $D'_n = \{\neg A\}$ then $F \vdash_{Res} A$ and $F \vdash_{Res} \neg A$

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \square$ from $F[\perp/A]$ and $D_0, \dots, D_n = \square$ from $F[\top/A]$.

Now transform C_0, \dots, C_m into a proof C'_0, \dots, C'_m from F by adding A back into the clauses it was removed from. Then

- ▶ either $C'_m = \{A\}$
- ▶ or $C'_m = \square$ (and we are done).

Similarly we transform D_0, \dots, D_n into a proof D'_0, \dots, D'_n from F (by adding $\neg A$ back in).

Then $D'_n = \{\neg A\}$ or $D'_n = \square$ (and we are done).

If $C'_m = \{A\}$ and $D'_n = \{\neg A\}$ then $F \vdash_{Res} A$ and $F \vdash_{Res} \neg A$ and thus $F \vdash_{Res} \square$.

Resolution is only refutation complete

Not everything that is a consequence of a set of clauses
can be derived by resolution.

Resolution is only refutation complete

Not everything that is a consequence of a set of clauses
can be derived by resolution.

Exercise

Find F and C such that $F \models C$ but not $F \vdash_{Res} C$.

Resolution is only refutation complete

Not everything that is a consequence of a set of clauses
can be derived by resolution.

Exercise

Find F and C such that $F \models C$ but not $F \vdash_{Res} C$.

How to prove $F \models C$ by resolution?

Resolution is only refutation complete

Not everything that is a consequence of a set of clauses
can be derived by resolution.

Exercise

Find F and C such that $F \models C$ but not $F \vdash_{Res} C$.

How to prove $F \models C$ by resolution?

Prove $F \cup \{\neg C\} \vdash_{Res} \square$

A resolution algorithm

Input: A CNF formula F , i.e. a finite set of clauses

A resolution algorithm

Input: A CNF formula F , i.e. a finite set of clauses

while there are clauses $C_a, C_b \in F$ and resolvent R of C_a and C_b
such that $R \notin F$

do $F := F \cup \{R\}$

A resolution algorithm

Input: A CNF formula F , i.e. a finite set of clauses

while there are clauses $C_a, C_b \in F$ and resolvent R of C_a and C_b
such that $R \notin F$

do $F := F \cup \{R\}$

Lemma

The algorithm terminates.

A resolution algorithm

Input: A CNF formula F , i.e. a finite set of clauses

while there are clauses $C_a, C_b \in F$ and resolvent R of C_a and C_b
such that $R \notin F$
do $F := F \cup \{R\}$

Lemma

The algorithm terminates.

Proof There are only finitely many clauses over a finite set of atoms.

A resolution algorithm

Input: A CNF formula F , i.e. a finite set of clauses

while there are clauses $C_a, C_b \in F$ and resolvent R of C_a and C_b
such that $R \notin F$
do $F := F \cup \{R\}$

Lemma

The algorithm terminates.

Proof There are only finitely many clauses over a finite set of atoms.

Theorem

The initial F is unsatisfiable iff \square is in the final F

A resolution algorithm

Input: A CNF formula F , i.e. a finite set of clauses

while there are clauses $C_a, C_b \in F$ and resolvent R of C_a and C_b
such that $R \notin F$
do $F := F \cup \{R\}$

Lemma

The algorithm terminates.

Proof There are only finitely many clauses over a finite set of atoms.

Theorem

The initial F is unsatisfiable iff \square is in the final F

Proof F_{init} is unsat. iff $F_{init} \vdash_{Res} \square$

A resolution algorithm

Input: A CNF formula F , i.e. a finite set of clauses

while there are clauses $C_a, C_b \in F$ and resolvent R of C_a and C_b
such that $R \notin F$
do $F := F \cup \{R\}$

Lemma

The algorithm terminates.

Proof There are only finitely many clauses over a finite set of atoms.

Theorem

The initial F is unsatisfiable iff \square is in the final F

Proof F_{init} is unsat. iff $F_{init} \vdash_{Res} \square$ iff $\square \in F_{final}$

A resolution algorithm

Input: A CNF formula F , i.e. a finite set of clauses

while there are clauses $C_a, C_b \in F$ and resolvent R of C_a and C_b
such that $R \notin F$
do $F := F \cup \{R\}$

Lemma

The algorithm terminates.

Proof There are only finitely many clauses over a finite set of atoms.

Theorem

The initial F is unsatisfiable iff \square is in the final F

Proof F_{init} is unsat. iff $F_{init} \vdash_{Res} \square$ iff $\square \in F_{final}$
because the algorithm enumerates all R such that $F_{init} \vdash R$.

A resolution algorithm

Input: A CNF formula F , i.e. a finite set of clauses

while there are clauses $C_a, C_b \in F$ and resolvent R of C_a and C_b
such that $R \notin F$
do $F := F \cup \{R\}$

Lemma

The algorithm terminates.

Proof There are only finitely many clauses over a finite set of atoms.

Theorem

The initial F is unsatisfiable iff \square is in the final F

Proof F_{init} is unsat. iff $F_{init} \vdash_{Res} \square$ iff $\square \in F_{final}$
because the algorithm enumerates all R such that $F_{init} \vdash R$.

Corollary

The algorithm is a decision procedure for unsatisfiability of CNF formulas.

Propositional Logic
CDCL: Conflict Driven Clause
Learning

CDCL: goal and idea

Goal: Combine DPLL and resolution into an algorithm oriented towards both satisfiability and unsatisfiability.

Idea: At every unsuccessful leaf of DPLL (called **conflict**), compute a **conflict clause**, and add it to the formula we are deciding about.

Conflict clauses “cache” previous search results, so we “learn from previous mistakes”.

Conflict clauses also determine backtracking.

We present a particular way of computing a conflict clause using resolution. There are other ways.

DPLL + CDCL algorithm

Input: CNF formula F .

1. Initialise \mathcal{A} to the empty assignment
2. While there is unit clause $\{L\}$ or pure literal L in $F|_{\mathcal{A}}$, update $\mathcal{A} \mapsto \mathcal{A}_{[L \mapsto 1]}$
3. If $F|_{\mathcal{A}}$ contains no clauses, stop and output \mathcal{A} .
4. If $F|_{\mathcal{A}} \ni \square$, add new clause C to F by **learning procedure**.
If C is the empty clause, stop and output UNSAT.
Otherwise backtrack to highest level where C is unit clause.
Go to Line 2.
5. Apply **decision strategy** to update $\mathcal{A} \mapsto \mathcal{A}_{[p \mapsto b]}$.
Go to line 2.

$F|_{\mathcal{A}}$ is set of clauses obtained from deleting any clause containing true literal, and deleting from each remaining clause all false literals.

Terminology

State of algorithm is pair of CNF formula F and assignment \mathcal{A} .
Successful state when $\mathcal{A} \models F$. **Conflict state** when $\mathcal{A} \not\models F$.

Terminology

State of algorithm is pair of CNF formula F and assignment \mathcal{A} .
Successful state when $\mathcal{A} \models F$. **Conflict state** when $\mathcal{A} \not\models F$.

- ▶ Each assignment $p_i \mapsto b_i$ classifies as **decision assignment** or **implied assignment**.

Terminology

State of algorithm is pair of CNF formula F and assignment \mathcal{A} .
Successful state when $\mathcal{A} \models F$. **Conflict state** when $\mathcal{A} \not\models F$.

- ▶ Each assignment $p_i \mapsto b_i$ classifies as **decision assignment** or **implied assignment**.
- ▶ p_i in a decision assignment $p_i \mapsto b_i$ is **decision variable**.

Terminology

State of algorithm is pair of CNF formula F and assignment \mathcal{A} .
Successful state when $\mathcal{A} \models F$. **Conflict state** when $\mathcal{A} \not\models F$.

- ▶ Each assignment $p_i \mapsto b_i$ classifies as **decision assignment** or **implied assignment**.
- ▶ p_i in a decision assignment $p_i \mapsto b_i$ is **decision variable**.
- ▶ Denote by $p_i \xrightarrow{C} b_i$ an implied assignment arising through **unit propagation** on clause C .

Terminology

State of algorithm is pair of CNF formula F and assignment \mathcal{A} .
Successful state when $\mathcal{A} \models F$. **Conflict state** when $\mathcal{A} \not\models F$.

- ▶ Each assignment $p_i \mapsto b_i$ classifies as **decision assignment** or **implied assignment**.
- ▶ p_i in a decision assignment $p_i \mapsto b_i$ is **decision variable**.
- ▶ Denote by $p_i \xrightarrow{C} b_i$ an implied assignment arising through **unit propagation** on clause C .
- ▶ **Decision level** of assignment $p_i \mapsto b_i$ in a given state \mathcal{A} is number of decision assignments in \mathcal{A} that precede $p_i \mapsto b_i$.

Terminology

State of algorithm is pair of CNF formula F and assignment \mathcal{A} .
Successful state when $\mathcal{A} \models F$. **Conflict state** when $\mathcal{A} \not\models F$.

- ▶ Each assignment $p_i \mapsto b_i$ classifies as **decision assignment** or **implied assignment**.
- ▶ p_i in a decision assignment $p_i \mapsto b_i$ is **decision variable**.
- ▶ Denote by $p_i \xrightarrow{C} b_i$ an implied assignment arising through **unit propagation** on clause C .
- ▶ **Decision level** of assignment $p_i \mapsto b_i$ in a given state \mathcal{A} is number of decision assignments in \mathcal{A} that precede $p_i \mapsto b_i$.

(Note: conflict state if $F|_{\mathcal{A}} \ni \square$, successful state if $F|_{\mathcal{A}} = \emptyset$)

Example: start with set of clauses $F = \{C_1, \dots, C_5\}$, where

$$C_1 = \{\neg p_1, \neg p_4, p_5\}$$

$$C_2 = \{\neg p_1, p_6, \neg p_5\}$$

$$C_3 = \{\neg p_1, \neg p_6, p_7\}$$

$$C_4 = \{\neg p_1, \neg p_7, \neg p_5\}$$

$$C_5 = \{p_1, p_4, p_6\}$$

Say current assignment is $(p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1)$.

Notice $F|_{\mathcal{A}}$ contains unit clause $\{p_5\}$.

Unit propagation further generates $(p_5 \xrightarrow{C_1} 1, p_6 \xrightarrow{C_2} 1, p_7 \xrightarrow{C_3} 1)$. This leads to a conflict, with C_4 being made false.

Conflict analysis

After unit propagation:

- ▶ If not in conflict nor successful, make decision (line 5)
- ▶ If in conflict, **learned clause** is added (line 4)

Conflict analysis

After unit propagation:

- ▶ If not in conflict nor successful, make decision (line 5)
- ▶ If in conflict, **learned clause** is added (line 4)

Learned clause desiderata: If unit propagation from state (F, \mathcal{A}) leads to conflict, clause C is learned such that:

Conflict analysis

After unit propagation:

- ▶ If not in conflict nor successful, make decision (line 5)
- ▶ If in conflict, **learned clause** is added (line 4)

Learned clause desiderata: If unit propagation from state (F, \mathcal{A}) leads to conflict, clause C is learned such that:

1. $F \equiv F \cup \{C\}$

Conflict analysis

After unit propagation:

- ▶ If not in conflict nor successful, make decision (line 5)
- ▶ If in conflict, **learned clause** is added (line 4)

Learned clause desiderata: If unit propagation from state (F, \mathcal{A}) leads to conflict, clause C is learned such that:

1. $F \equiv F \cup \{C\}$
2. C is **conflict clause**: each literal is made false by \mathcal{A}

Conflict analysis

After unit propagation:

- ▶ If not in conflict nor successful, make decision (line 5)
- ▶ If in conflict, **learned clause** is added (line 4)

Learned clause desiderata: If unit propagation from state (F, \mathcal{A}) leads to conflict, clause C is learned such that:

1. $F \equiv F \cup \{C\}$
2. C is **conflict clause**: each literal is made false by \mathcal{A}
3. C mentions only decision variables in \mathcal{A}

Clause learning using resolution

Suppose $\mathcal{A} = (p_1 \mapsto b_1, \dots, p_k \mapsto b_k)$ leads to conflict.

Find associated clauses A_1, \dots, A_{k+1} by backward induction:

Clause learning using resolution

Suppose $\mathcal{A} = (p_1 \mapsto b_1, \dots, p_k \mapsto b_k)$ leads to conflict.

Find associated clauses A_1, \dots, A_{k+1} by backward induction:

1. Take any conflict clause under \mathcal{A} as A_{k+1} .

Clause learning using resolution

Suppose $\mathcal{A} = (p_1 \mapsto b_1, \dots, p_k \mapsto b_k)$ leads to conflict.

Find associated clauses A_1, \dots, A_{k+1} by backward induction:

1. Take any conflict clause under \mathcal{A} as A_{k+1} .
2. If $p_i \mapsto b_i$ is decision assignment or p_i not mentioned in A_{i+1} , set $A_i = A_{i+1}$.

Clause learning using resolution

Suppose $\mathcal{A} = (p_1 \mapsto b_1, \dots, p_k \mapsto b_k)$ leads to conflict.

Find associated clauses A_1, \dots, A_{k+1} by backward induction:

1. Take any conflict clause under \mathcal{A} as A_{k+1} .
2. If $p_i \mapsto b_i$ is decision assignment or p_i not mentioned in A_{i+1} , set $A_i = A_{i+1}$.
3. If $p_i \stackrel{C_i}{\mapsto} b_i$ is implied assignment and p_i mentioned in A_{i+1} , define A_i to be resolvent of A_{i+1} and C_i with respect to p_i .

Clause learning using resolution

Suppose $\mathcal{A} = (p_1 \mapsto b_1, \dots, p_k \mapsto b_k)$ leads to conflict.

Find associated clauses A_1, \dots, A_{k+1} by backward induction:

1. Take any conflict clause under \mathcal{A} as A_{k+1} .
2. If $p_i \mapsto b_i$ is decision assignment or p_i not mentioned in A_{i+1} , set $A_i = A_{i+1}$.
3. If $p_i \stackrel{C_i}{\mapsto} b_i$ is implied assignment and p_i mentioned in A_{i+1} , define A_i to be resolvent of A_{i+1} and C_i with respect to p_i .

$C := A_1$, that is, the final clause A_1 is the **learned clause** .

Clause learning: example

In conflict of above example, learning generates clauses

$$A_8 := \{\neg p_1, \neg p_7, \neg p_5\} \quad (\text{clause } C_4)$$

$$A_7 := \{\neg p_1, \neg p_5, \neg p_6\} \quad (\text{resolve } A_8, C_3)$$

$$A_6 := \{\neg p_1, \neg p_5\} \quad (\text{resolve } A_7, C_2)$$

$$A_5 := \{\neg p_1, \neg p_4\} \quad (\text{resolve } A_6, C_1)$$

⋮

$$A_1 := \{\neg p_1, \neg p_4\}$$

Clause learning: example

In conflict of above example, learning generates clauses

$$A_8 := \{\neg p_1, \neg p_7, \neg p_5\} \quad (\text{clause } C_4)$$

$$A_7 := \{\neg p_1, \neg p_5, \neg p_6\} \quad (\text{resolve } A_8, C_3)$$

$$A_6 := \{\neg p_1, \neg p_5\} \quad (\text{resolve } A_7, C_2)$$

$$A_5 := \{\neg p_1, \neg p_4\} \quad (\text{resolve } A_6, C_1)$$

⋮

$$A_1 := \{\neg p_1, \neg p_4\}$$

Learned clause A_1 is conflict clause with only decision variables, including top-level one p_4 . Intuitively:

- ▶ A_1 records that conflict arose from decision to make p_1, p_4 true.
- ▶ Adding A_1 makes assignments setting $p_1, p_4 \mapsto 1$ unreachable
- ▶ DPLL backtracks to highest level where A_1 is unit clause ($p_1 \mapsto 1$), unit propagation leads to $p_4 \mapsto 0$.

Clause learning

Proposition: The clause learning procedure satisfies the three desiderata.

Proof sketch: **Observation:** If $p_i \stackrel{C_i}{\vdash} b_i$, then the only literal of C_i true under \mathcal{A} is the literal for p_i (that is, C_i contains either p_i or $\neg p_i$, and b_i is chosen to make the literal true).

1. $F \equiv F \cup \{C\}$

Because C is obtained from clauses of F through resolution steps.

Clause learning

Proposition: The clause learning procedure satisfies the three desiderata.

Proof sketch: **Observation:** If $p_i \stackrel{C_i}{\mapsto} b_i$, then the only literal of C_i true under \mathcal{A} is the literal for p_i (that is, C_i contains either p_i or $\neg p_i$, and b_i is chosen to make the literal true).

1. $F \equiv F \cup \{C\}$

Because C is obtained from clauses of F through resolution steps.

2. C is **conflict clause**: each literal is made false by \mathcal{A} .

We show by induction that $A_{k+1}, A_k, \dots, A_1 = C$ are conflict clauses.

A_{k+1} is conflict clause by definition. If A_{i+1} is conflict clause and $A_i = A_{i+1}$, then so is A_i .

If A_{i+1} is conflict clause and $A_i \neq A_{i+1}$, then A_i is the result of resolving A_{i+1} and C_i . By the **observation**, all literals of A_i are made false by \mathcal{A} .

Clause learning

Proposition: The clause learning procedure satisfies the three desiderata.

Proof sketch: **Observation:** If $p_i \stackrel{C_i}{\mapsto} b_i$, then the only literal of C_i true under \mathcal{A} is the literal for p_i (that is, C_i contains either p_i or $\neg p_i$, and b_i is chosen to make the literal true).

1. $F \equiv F \cup \{C\}$

Because C is obtained from clauses of F through resolution steps.

2. C is **conflict clause**: each literal is made false by \mathcal{A} .

We show by induction that $A_{k+1}, A_k, \dots, A_1 = C$ are conflict clauses.

A_{k+1} is conflict clause by definition. If A_{i+1} is conflict clause and $A_i = A_{i+1}$, then so is A_i .

If A_{i+1} is conflict clause and $A_i \neq A_{i+1}$, then A_i is the result of resolving A_{i+1} and C_i . By the **observation**, all literals of A_i are made false by \mathcal{A} .

3. C mentions only decision variables in \mathcal{A} .

Because every other variable, say p_i , disappears after resolving with A_{i+1} w.r.t. p_i . Indeed, since \mathcal{A} makes A_{i+1} false, by the **observation** p_i has opposite signs in A_{i+1} and C_i .

Example (without PLR)

$$\{\neg p_1\} \{p_1, p_3, p_4\} \{\neg p_2, \neg p_5\} \{p_3, \neg p_4, p_5, \neg p_6\} \{p_1, \neg p_2, \neg p_4, p_6\}$$

UP:	$p_1 \mapsto 0$	$\{p_3, p_4\}$	$\{\neg p_2, \neg p_5\}$	$\{p_3, \neg p_4, p_5, \neg p_6\}$	$\{\neg p_2, \neg p_4, p_6\}$
DE:	$p_2 \mapsto 1$	$\{p_3, p_4\}$	$\{\neg p_5\}$	$\{p_3, \neg p_4, p_5, \neg p_6\}$	$\{\neg p_4, p_6\}$
UP:	$p_5 \mapsto 0$	$\{p_3, p_4\}$		$\{p_3, \neg p_4, \neg p_6\}$	$\{\neg p_4, p_6\}$
DE:	$p_3 \mapsto 0$	$\{p_4\}$		$\{\neg p_4, \neg p_6\}$	$\{\neg p_4, p_6\}$
UP:	$p_4 \mapsto 1$			$\{\neg p_6\}$	$\{p_6\}$
UP:	$p_6 \mapsto 1$			$\{\}$	

$$A_7 := \{p_3, \neg p_4, p_5, \neg p_6\} \quad (\text{conflict clause})$$

$$A_6 := \{p_1, \neg p_2, p_3, \neg p_4, p_5\} \quad (\text{resolve } A_7, \{p_1, \neg p_2, \neg p_4, p_6\})$$

$$A_5 := \{p_1, \neg p_2, p_3, p_5\} \quad (\text{resolve } A_6, \{p_1, p_3, p_4\})$$

$$A_4 := A_5$$

$$A_3 := \{p_1, \neg p_2, p_3\} \quad (\text{resolve } A_4, \{p_1, p_3, p_4\})$$

$$A_2 := A_3$$

$$A_1 := \{\neg p_2, p_3\} \quad (\text{resolve } A_2, \{\neg p_1\})$$

Backtracking to $\{p_1 \mapsto 0, p_2 \mapsto 1\}$. Unit propagation: $p_3 \mapsto 1$.