Propositional Logic
Basics

Syntax of propositional logic

Definition

An atomic formula (or atom) has the form A_i where $i = 1, 2, 3, \ldots$.
Formulas are defined inductively:

- $\bot$ ("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 $A \rightarrow B \wedge \neg(C \vee D) \vee E$.

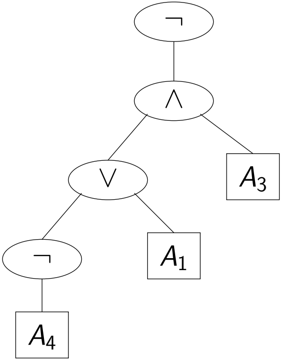
Outermost parentheses can be dropped.

Syntax tree of a formula

Every formula can be represented by a syntax tree.

Example

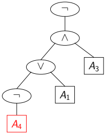
$F = \neg((\neg A_4 \vee A_1) \wedge A_3)$



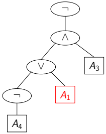
Subformulas

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

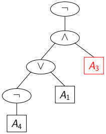
A_4



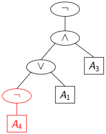
A_1



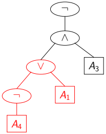
A_3



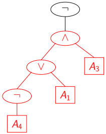
$\neg A_4$



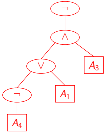
$(\neg A_4 \lor A_1)$



$((\neg A_4 \lor A_1) \land A_3)$



$\neg((\neg A_4 \lor A_1) \land A_3)$



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 $\mathcal{P}(\bot)$, 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 $\mathcal{A} : \mathit{Atoms} \to \{0, 1\}$
where *Atoms* is the set of all atoms.

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

Semantics of propositional logic (II)

$$
\begin{array}{rcl}
\hat{\mathcal{A}}(A_i) & = & \mathcal{A}(A_i) \\
\hat{\mathcal{A}}(\neg F) & = & \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 0 \\ 0 & \text{otherwise} \end{cases} \\
\hat{\mathcal{A}}(F \wedge G) & = & \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 1 \text{ and } \hat{\mathcal{A}}(G) = 1 \\ 0 & \text{otherwise} \end{cases} \\
\hat{\mathcal{A}}(F \vee G) & = & \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 1 \text{ or } \hat{\mathcal{A}}(G) = 1 \\ 0 & \text{otherwise} \end{cases} \\
\hat{\mathcal{A}}(F \rightarrow G) & = & \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 0 \text{ or } \hat{\mathcal{A}}(G) = 1 \\ 0 & \text{otherwise} \end{cases}
\end{array}
$$

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

$$\begin{array}{lll} A, B, C, & & \\ P, Q, R, \text{ or } \ldots & \text{instead of} & A_1, A_2, A_3 \ldots \end{array}$$

$$\begin{array}{rll} F_1 \leftrightarrow F_2 & \text{abbreviates} & (F_1 \land F_2) \lor (\neg F_1 \land \neg F_2) \\ \bigvee_{i=1}^{n} F_i & \text{abbreviates} & (\ldots((F_1 \lor F_2) \lor F_3) \lor \ldots \lor F_n) \\ \bigwedge_{i=1}^{n} F_i & \text{abbreviates} & (\ldots((F_1 \land F_2) \land F_3) \land \ldots \land F_n) \end{array}$$

Special cases:

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

Truth tables (I)

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

$\neg$	A
1	0
0	1

A	$\vee$	B
0	0	0
0	1	1
1	1	0
1	1	1

A	$\wedge$	B
0	0	0
0	0	1
1	0	0
1	1	1

A	$\rightarrow$	B
0	1	0
0	1	1
1	0	0
1	1	1

A	$\leftrightarrow$	B
0	1	0
0	0	1
1	0	0
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. $\square$

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 assigment that is a model of every formula in S.

Exercise

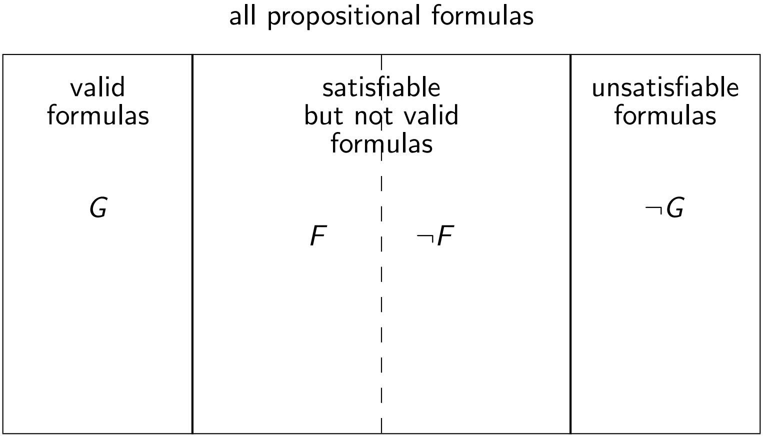
	Valid	Satisfiable	Unsatisfiable
A			
$A \vee B$			
$A \vee \neg A$			
$A \wedge \neg A$			
$A \rightarrow \neg A$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Exercise

Which of the following statements are true?

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

Mirroring principle



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 $\mathcal{A} \models (R \wedge \neg A) \wedge B$.
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 $\mathcal{A} \models A \rightarrow B$
- $A \models \neg A$ follows from $\mathcal{A} \models B$ and $\mathcal{A} \models R$.

Exercise

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

Consequence

Exercise

The following statements are equivalent:

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

Proof of "if $F_1, \ldots, F_k \models G$ then $\models \underbrace{(\bigwedge_{i=1}^{k} F_i) \to G}_{H}$".

Assume $F_1, \ldots, F_k \models G$.
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 $H = \bigwedge F_i \to G$.
- $\mathcal{A}(\bigwedge F_i) = 1$: Then $\mathcal{A}(F_i) = 1$ for all i.
  Therefore $\mathcal{A}$ is a model of $F_1, \ldots, F_k$.
  Therefore $\mathcal{A} \models G$ because $F_1, \ldots, F_k \models G$.

Validity and satisfiability

Exercise

The following statements are equivalent:

1. $F \to G$ *is valid.*
2. $F \land \neg G$ *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 $M \models F$.		
If F valid then $M \models F$.		
If $F \in M$ then $M \models F$.		
If $F \models G$ then $\neg F \models \neg G$.		

Notation

Warning: The symbol $\models$ is overloaded:

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

$$\models F$$

$$M \models F$$

Convenient variations for set of formulas S:

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

$\models S$ means that for all $F \in S$, $\models F$

$M \models S$ 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?

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

Observation

The following connections hold:

$$\models F \to G \quad \text{iff} \quad F \models G$$
$$\models F \leftrightarrow G \quad \text{iff} \quad F \equiv G$$

NB: “iff” means “if and only if”

Reductions between problems (I)

- Validity to Unsatisfiability:

$$F \text{ valid} \quad \text{iff} \quad \textbf{?} \neg F \text{ unsatisfiable}$$

- Unsatisfiability to Validity:

$$F \text{ unsatisfiable} \quad \text{iff} \quad \textbf{?} \; \neg F \text{ valid}$$

- Validity to Consequence:

$$F \text{ valid} \quad \text{iff} \quad \textbf{?} \models \textbf{?} \; \top \models F$$

- Consequence to Validity:

$$F \models G \quad \text{iff} \quad \textbf{?} \; F \to G \text{ valid}$$

- Validity to Equivalence:

$$F \text{ valid} \quad \text{iff} \quad \textbf{?} \equiv \textbf{?} \; F \equiv \top$$

- Equivalence to Validity:

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

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 $\circ \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$

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 $H \equiv H'$.

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

$$
\begin{array}{rcll}
(F \wedge F) & \equiv & F & \\
(F \vee F) & \equiv & F & \textit{(Idempotence)} \\
(F \wedge G) & \equiv & (G \wedge F) & \\
(F \vee G) & \equiv & (G \vee F) & \textit{(Commutativity)} \\
((F \wedge G) \wedge H) & \equiv & (F \wedge (G \wedge H)) & \\
((F \vee G) \vee H) & \equiv & (F \vee (G \vee H)) & \textit{(Associativity)} \\
(F \wedge (F \vee G)) & \equiv & F & \\
(F \vee (F \wedge G)) & \equiv & F & \textit{(Absorption)}
\end{array}
$$

Equivalences (II)

$$
\begin{array}{rcll}
(F \wedge (G \vee H)) & \equiv & ((F \wedge G) \vee (F \wedge H)) & \\
(F \vee (G \wedge H)) & \equiv & ((F \vee G) \wedge (F \vee H)) & \text{(Distributivity)} \\
\neg\neg F & \equiv & F & \text{(Double negation)} \\
\neg(F \wedge G) & \equiv & (\neg F \vee \neg G) & \\
\neg(F \vee G) & \equiv & (\neg F \wedge \neg G) & \text{(deMorgan's Laws)} \\
\neg\top & \equiv & \bot & \\
\neg\bot & \equiv & \top & \\
(\top \vee G) & \equiv & \top & \\
(\top \wedge G) & \equiv & G & \\
(\bot \vee G) & \equiv & G & \\
(\bot \wedge G) & \equiv & \bot &
\end{array}
$$

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:

$$F_1 \rightarrow F_2 \quad \text{abbreviates} \quad \neg F_1 \vee F_2$$

$$F_1 \leftrightarrow F_2 \quad \text{abbreviates} \quad (F_1 \wedge F_2) \vee (\neg F_1 \wedge \neg F_2)$$

$$\top \quad \text{abbreviates} \quad A_1 \vee \neg A_1$$

$$\bot \quad \text{abbreviates} \quad A_1 \wedge \neg A_1$$

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 formal 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:

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

Example

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

("$F \equiv G \equiv H$" is an abbreviation for "$F \equiv G$ and $G \equiv H$")

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:

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

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

Definition

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

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

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

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:

$$\begin{array}{rcl} (F \vee (G \wedge H)) & \equiv & ((F \vee G) \wedge (F \vee H)) \\ ((F \wedge G) \vee H) & \equiv & ((F \vee H) \wedge (G \vee H)) \end{array}$$

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

$$\begin{array}{rcl} (F \wedge (G \vee H)) & \equiv & ((F \wedge G) \vee (F \wedge H)) \\ ((F \vee G) \wedge H) & \equiv & ((F \wedge H) \vee (G \wedge H)) \end{array}$$

Termination

Why does the transformation into NNF and CNF terminate?

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

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

Define w recursively:
$w(A_i) = \ldots$
$w(\neg F) = \ldots w(F) \ldots$
$w(F \wedge G) = \ldots w(F) \ldots w(G) \ldots$
$w(F \vee G) = \ldots w(F) \ldots w(G) \ldots$

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 instist 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

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

   Repeatedly replace a subformula G of the form $\neg A$, $A \wedge B$ or $A \vee B$ (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

$$
\begin{array}{ll}
 & \neg(\boxed{A_1 \vee A_2}) \wedge A_3 \\
\rightsquigarrow & \boxed{\neg A_4} \wedge A_3 \wedge (A_4 \leftrightarrow (A_1 \vee A_2)) \\
\rightsquigarrow & \boxed{A_5 \wedge A_3} \wedge (A_4 \leftrightarrow (A_1 \vee A_2)) \wedge (A_5 \leftrightarrow \neg A_4) \\
\rightsquigarrow & A_6 \wedge (A_4 \leftrightarrow (A_1 \vee A_2)) \wedge (A_5 \leftrightarrow \neg A_4) \wedge (A_6 \leftrightarrow (A_5 \wedge A_3)) \\
\rightsquigarrow & A_6 \wedge CNF(A_4 \leftrightarrow (A_1 \vee A_2)) \wedge CNF(A_5 \leftrightarrow \neg A_4) \wedge CNF(A_6 \leftrightarrow (A_5 \wedge A_3))
\end{array}
$$

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 $O(n)$, and
can be constructed in $O(n)$ time.

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

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

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
$\mathcal{A}(F[G/A]) \;=\; \mathcal{A}'(F)$ *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 $F \neq A$: $\mathcal{A}(F[G/A]) = \mathcal{A}(F) = \mathcal{A}'(F)$
- $F = F_1 \wedge F_2$:

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

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 $F \wedge (A \leftrightarrow G)$.

Proof Assume $\mathcal{A} \models F[G/A]$ for some assignment $\mathcal{A}$.
Let $\mathcal{A}' := \mathcal{A}[\mathcal{A}(G)/A]$. We prove $\mathcal{A}' \models F \wedge (A \leftrightarrow G)$.
$\mathcal{A}' \models F$: Substitution Lemma.
$\mathcal{A}' \models (A \leftrightarrow G)$: Because $\mathcal{A}'(A) = \mathcal{A}(G) = \mathcal{A}'(G)$
(by definition of $\mathcal{A}'$ and because A does not occur in G).

Assume $\mathcal{A} \models F \wedge (A \leftrightarrow G)$ for some assignment $\mathcal{A}$.
We prove $\mathcal{A} \models F[G/A]$, that is, $\mathcal{A}(F[G/A]) = 1$.
We show $\mathcal{A}(F[G/A]) = \mathcal{A}'(F) = \mathcal{A}(F) = 1$ for $\mathcal{A}' := \mathcal{A}[\mathcal{A}(G)/A]$.
$\mathcal{A}(F[G/A]) = \mathcal{A}'(F)$: Substitution Lemma.
$\mathcal{A}'(F) = \mathcal{A}(F)$: From $\mathcal{A} \models (A \leftrightarrow G)$ follows $\mathcal{A}(A) = \mathcal{A}(G)$, and so $\mathcal{A}' = \mathcal{A}$.
$\mathcal{A}(F) = 1$: Because $\mathcal{A} \models F$.

Definitional CNF: Correctness

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 $O(n)$.

Proof.

Repeated application of the Lemma. □

Similarly it can be shown:

Theorem

For every formula F of size n there is an equivalid 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: $(A \lor \neg A \lor B) \land (C \lor \neg C)$

Not valid: $(A \lor \neg A) \land (\neg A \lor C)$

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: $(\neg B \land A \land B) \lor (\neg A \land C)$

Unsatisfiable: $(A \land \neg A \land B) \lor (C \land \neg C)$

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 this and the next slide sets:

- A very efficient satisfiability check for a special class of formulas in CNF: Horn formulas,
- Efficient satisfiability checks for arbitrary formulas in CNF: DPLL and 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.

Every disjunct of a Horn formula can equivalently be viewed as an implication $K \to B$ where

- K is a conjunction of atoms or $\top$, and
- B is an atom or $\bot$.

$$
\begin{array}{rcll}
A & \equiv & (\top \to A) & \text{fact} \\
(\neg A \vee \neg B \vee C) & \equiv & (A \wedge B \to C) & \text{rule} \\
(\neg A \vee B) & \equiv & (A \to B) & \text{rule} \\
\neg A & \equiv & (A \to \bot) & \text{goal} \\
(\neg A \vee \neg B) & \equiv & (A \wedge B \to \bot) & \text{goal}
\end{array}
$$

Satisfiability check for Horn formulas

Input: a Horn formula F.
Output: Model $\mathcal{M}$ of F or "unsatisfiable"

for all atoms A_i in F **do** $\mathcal{M}(A_i) := 0$;
while F has a conjunct $K \to B$
such that $\mathcal{M}(K) = 1$ and $\mathcal{M}(B) = 0$
do
if $B = \perp$ **then return** "unsatisfiable"
else $\mathcal{M}(B) := 1$

return $\mathcal{M}$

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 Schöning.]

Correctness of the model building algorithm

Theorem

The algorithm returns a model iff F is satisfiable.

Proof. Invariant: if $\mathcal{M}(A) = 1$, then $\mathcal{A}(A) = 1$ for every atom A and model $\mathcal{A}$ of F.

(a) If "unsatisfiable" then unsatisfiable.
Assume F has model $\mathcal{A}$ but algorithm answers "unsatisfiable".
Let $(A_{i_1} \wedge \ldots \wedge A_{i_k} \rightarrow \bot)$ be the subformula causing "unsatisfiable".
Since $\mathcal{M}(A_{i_1}) = \cdots = \mathcal{M}(A_{i_k}) = 1$, $\mathcal{A}(A_{i_1}) = \ldots = \mathcal{A}(A_{i_k}) = 1$.
Then $\mathcal{A}(A_{i_1} \wedge \ldots \wedge A_{i_k} \rightarrow \bot) = 0$ and so $\mathcal{A}(F) = 0$, contradiction.

(b) If "$\mathcal{M}$" then $\mathcal{M} \models F$.
After termination with "$\mathcal{M}$", every conjunct $K \rightarrow B$ of F satisfies $\mathcal{M}(K) = 0$ or $\mathcal{M}(B) = 1$.
Therefore $\mathcal{M}(K \rightarrow B) = 1$ and thus $\mathcal{M} \models F$.

Correctness of the model building algorithm

Corollary

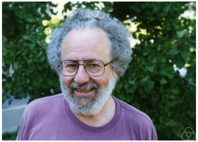
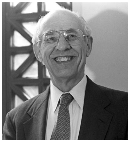
A satisfiable Horn formula has a unique model with a smallest number of true atoms.

Propositional Logic
DPLL: Davis-Putnam-Logemann-Loveland

Davis–Putnam–Logemann–Loveland

DPLL algorithm:

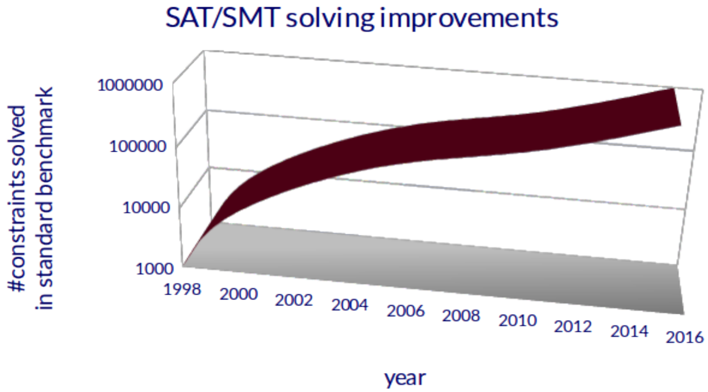
- combines search and deduction to decide satisfiability
- underlies most modern SAT solvers
- is over 50 years old

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

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

Representation as set of sets of literals:

$$\{\underbrace{\{L_{1,1}, \ldots, L_{1,n_1}\}}_{\textit{clause}}, \ldots, \{L_{k,1}, \ldots, L_{1,n_k}\}\}$$

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":

- 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 with built in associativity, commutativity and idempotence

CNF-SAT: Input: Set of clauses F
Question: Is F unsatisfiable?

DPLL — The simplest algorithm for CNF-SAT

Simplest algorithm: Construct the truth table.
Best-case runtime is $\Theta(m \cdot 2^n)$ for a formula of length m over n variables.

DPLL — A first improvement: Partial Evaluation

Improvement: partial evaluation using Boole-Shannon expansion

Lemma (Boole-Shannon Expansion)

For every formula F and atom A:

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

Proof By structural induction on F (exercise).

Corollary

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

DPLL — First step: partial evaluation

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

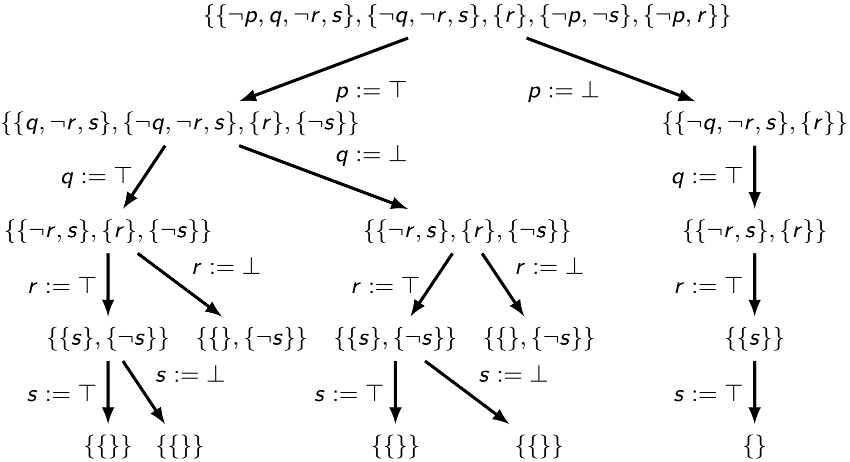
$F[\top/A] \equiv$ take F, remove all clauses with A, remove all $\neg A$.
$F[\bot/A] \equiv$ take F, remove all clauses with $\neg A$, remove all A.

Partial evaluation algorithm:

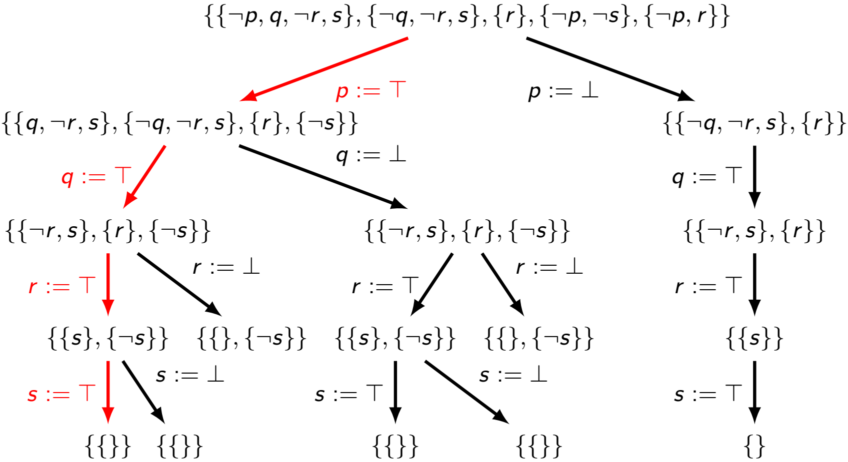
Given formula F, total order on the variables $\prec$:
If $\{\} \in F$ return unsatisfiable.
If $F = \emptyset$ return satisfiable.
Otherwise:

Fix the first variable A in F according to $\prec$.
Recursively check if $F[\bot/A]$ is satisfiable;
if yes, return satisfiable.
Recursively check if $F[\top/A]$ is satisfiable;
if yes, return satisfiable, otherwise unsatisfiable.

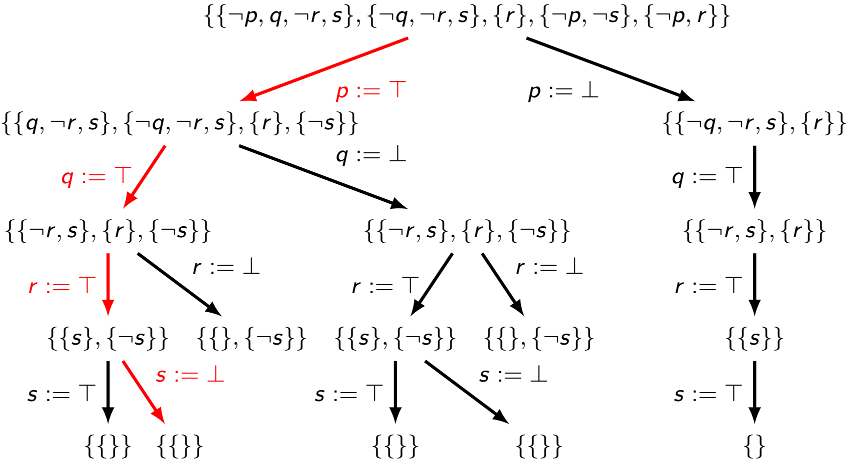
DPLL: Davis-Putnam-Logemann-Loveland



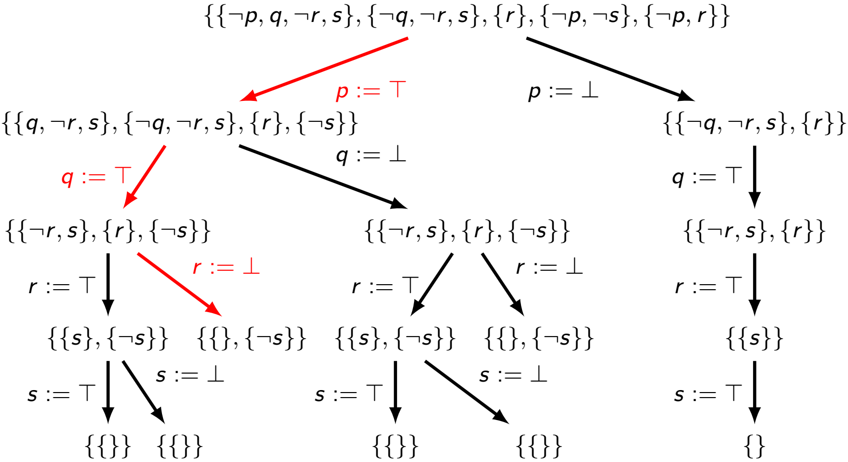
DPLL: Davis-Putnam-Logemann-Loveland



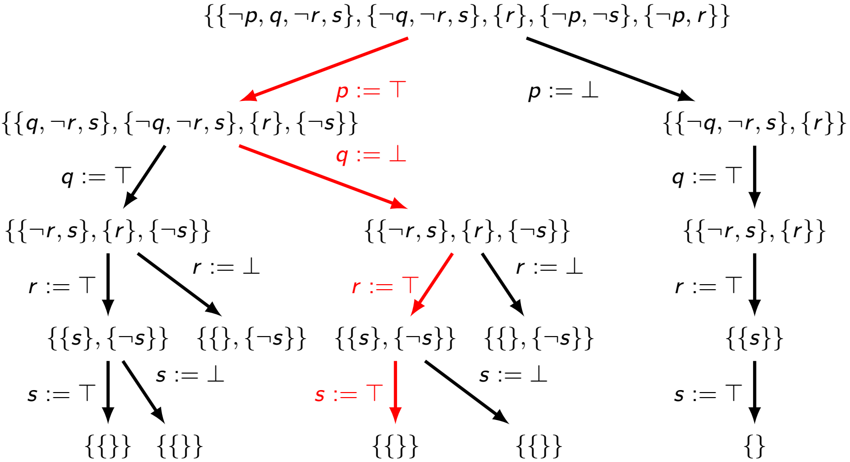
DPLL: Davis-Putnam-Logemann-Loveland



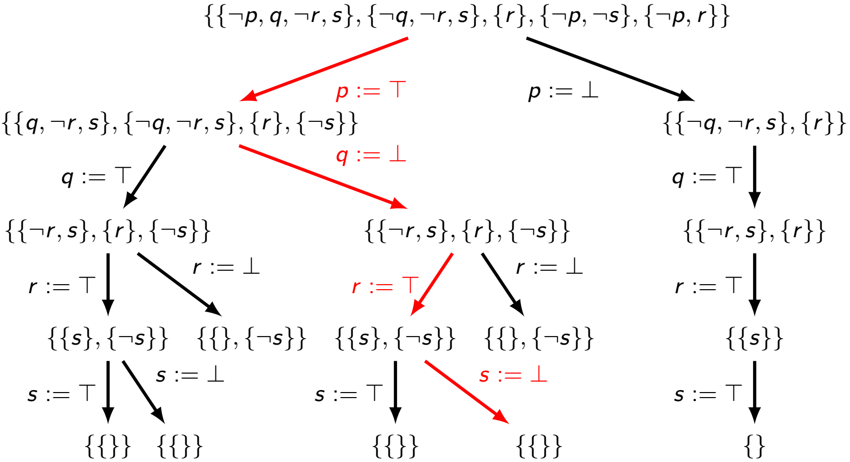
DPLL: Davis-Putnam-Logemann-Loveland



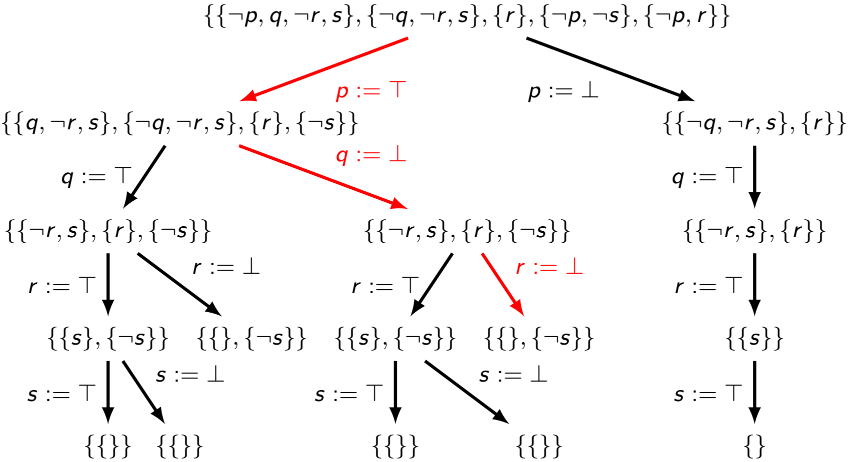
DPLL: Davis-Putnam-Logemann-Loveland



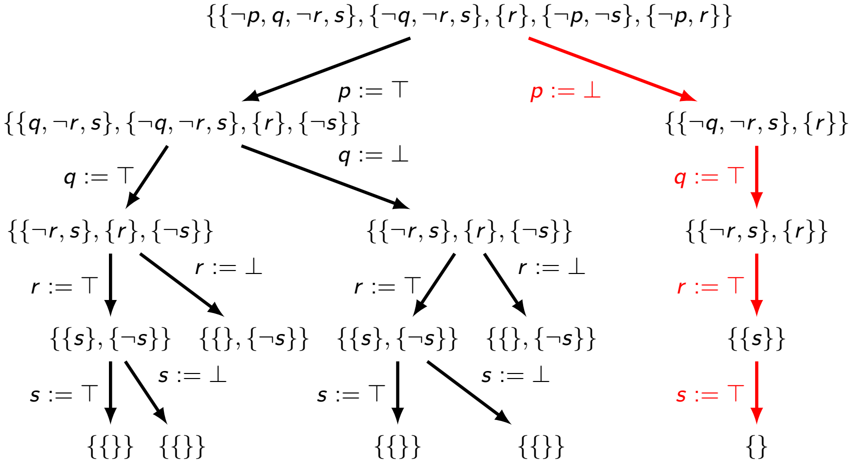
DPLL: Davis-Putnam-Logemann-Loveland



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 $\overline{L}$ 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

$$\{\{\neg p, q, \neg r, s\}, \{\neg q, \neg r, s\}, \{r\}, \{\neg p, \neg s\}, \{\neg p, r\}\}$$

$$\Big\downarrow \text{ OLR } r := \top$$

$$\{\{\neg p, q, s\}, \{\neg q, s\}, \{\neg p, \neg s\}\}$$

$$\Big\downarrow \text{ PLR } p := \bot$$

$$\{\{\neg q, s\}\}$$

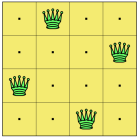
$$\Big\downarrow \text{ PLR } q := \bot$$

$$\{\}$$

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!

- 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 unsatisfiabiity.
- However, the certificate is exponentially longer than the formula in the worst case.
- Polynomial certificates for satisfiability implies NP= coNP.

Propositional Logic
Compactness

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 $c : V \to \{1, 2, 3, 4\}$ such that $(x, y) \in E$ implies $c(x) \neq c(y)$.

Theorem (4CT)

A finite planar graph has a 4-coloring.

Theorem

A planar graph $G = (V, E)$ with countably many vertices $V = \{v_1, v_2, \ldots\}$ 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

$G \rightsquigarrow S$:

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

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

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

$\Rightarrow$: 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

Definition

Let $b_1 \cdots b_n \in \{0,1\}^*$ with $n \geq 0$ and let T be a set of formulas. An assignment $\mathcal{A}$ is a $b_1 \cdots b_n$–model of T if $\mathcal{A}(A_i) = b_i$ for every $i = 1, \ldots, n$ and $\mathcal{A} \models T$.

In particular: every model is a ε-model.

Assume every finite $T \subseteq S$ is satisfiable. We prove:

1. There is an infinite sequence $b_1 b_2 \cdots \in \{0,1\}^\omega$ such that for every $n \geq 1$ all finite $T \subseteq S$ have a $b_1 \cdots b_n$–model.
2. The assignment $\mathcal{B}$ given by $\mathcal{B}(A_i) := b_i$ for all $i \geq 1$ is a model of S.

Proof of Compactness: Part (1)

To prove: There is an infinite sequence $b_1 b_2 \cdots \in \{0,1\}^\omega$ such that for every $n \geq 1$ all finite $T \subseteq S$ have a $b_1 \cdots b_n$–model.

It suffices to show:

(a) Every finite $T \subseteq S$ has an ε–model.

(b) For every sequence $\sigma \in \{0,1\}^*$: if every finite $T \subseteq S$ has a σ–model then there exists $b \in \{0,1\}$ such that every finite $T \subseteq S$ has a σb–model.

Proof of (a): By assumption every $T \subseteq S$ has an ε–model.

Proof of (b): Next slide.

Proof of Compactness: Part (1)

Proof of (b): By contradiction.

Assume that for some $\sigma \in \{0, 1\}^*$: every finite $T \subseteq S$ has a σ–model, but

(0) some finite $T_0 \subseteq S$ has no $\sigma 0$–model; and

(1) some finite $T_1 \subseteq S$ has no $\sigma 1$–model.

Consider the finite set $T_0 \cup T_1$.
By assumption, $T_0 \cup T_1$ has some σ–model $\mathcal{A}$. Let $n := |\sigma|$.

Two possible cases:

- $\mathcal{A}(A_{n+1}) = 0$. Then $\mathcal{A}$ is a $\sigma 0$–model of T_0, contradicting (0).
- $\mathcal{A}(A_{n+1}) = 1$. Then $\mathcal{A}$ is a $\sigma 1$–model of T_1, contradicting (1).

Proof of Compactness: Part (2)

To prove: The assignment $\mathcal{B}$ given by $\mathcal{B}(A_i) := b_i$ for all $i \geq 1$, where $b_1 b_2 \cdots$ is the infinite sequence of (1), is a model of S.

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

Let m be the maximal index of all atoms in F.
By (1), $\{F\}$ has a $b_1 \cdots b_m$–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
Resolution

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

Definition

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

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

Definition

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

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

is a resolvent of C_1 and C_2.
The process of deriving the resolvent is called a resolution step.

Graphical representation of resolvent:

$$\begin{tikzcd}
C_1 \arrow[dr, dash] & & C_2 \arrow[dl, dash] \\
& R &
\end{tikzcd}$$

If $C_1 = \{L\}$ and $C_2 = \{\overline{L}\}$ 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 $C_0, \ldots, C_n$ such that

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

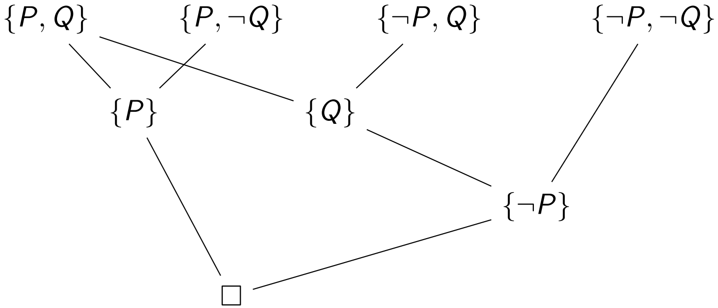
Then we can write $F \vdash_{Res} C$.

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:

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

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).
Assume $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$. We show $\mathcal{A} \models R$.
There are two cases:

- $\mathcal{A} \models L$. Then $\mathcal{A} \models C_2 - \{\bar{L}\}$ (because $\mathcal{A} \models C_2$), thus $\mathcal{A} \models R$.
- $\mathcal{A} \not\models L$. Then $\mathcal{A} \models C_1 - \{L\}$ (because $\mathcal{A} \models C_1$), thus $\mathcal{A} \models R$.

Correctness of resolution

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 $C_0, \ldots, C_n = C$.
We show $F \models C_i$ by induction on i. IH: $F \models C_j$ for all $j < i$.
There are two cases:

- $C_i \in F$.
  Then $F \models C_i$ by definition.
- C_i is a resolvent of C_a and C_b for $a, b < i$.
  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 $F \vdash_{Res} \Box$ then F is unsatisfiable.

Completeness of resolution

Theorem

Let F be a finite set of clauses. If F is unsatisfiable then $F \vdash_{Res} \Box$.

Theorem (Completeness of resolution)

Let F be a set of clauses. If F is unsatisfiable then $F \vdash_{Res} \Box$.

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 $F \vdash_{Res} \Box$.

Completeness proof

Corollary

(of the Boole-Shannon expansion) F is unsatisfiable iff $F[\bot/A]$ and $F[\top/A]$ are unsatisfiable.

Idea for completeness proof:
If A is an atom of F, then both $F[\bot/A]$ and $F[\top/A]$ have fewer atoms than F.

Use Boole-Shannon to prove completeness by induction on the number of atoms of the unsatisfiable formula F:

- construct inductively resolution proofs for $F[\bot/A]$ 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 $F[\bot/s]$.

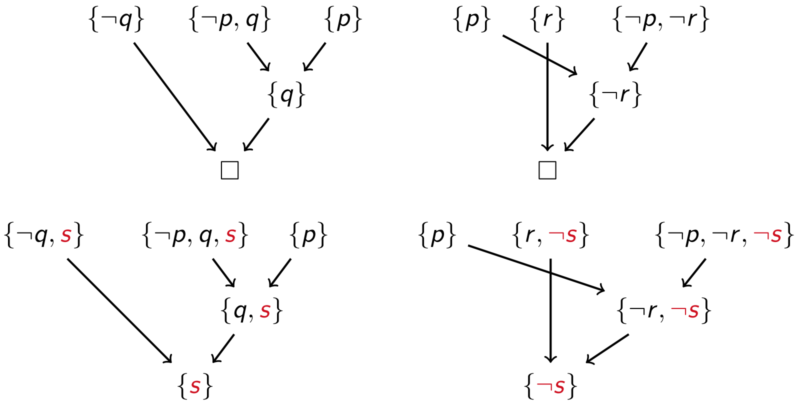
$$\begin{aligned} F[\top/s] &\equiv \{\ \{p\}\ ,\ \{r\}\ ,\ \{\neg p, \neg r\}\ \} \\ F[\bot/s] &\equiv \{\ \{\neg q\}\ ,\ \{\neg p, q\}\ ,\ \{p\}\ \} \end{aligned}$$

$$\begin{tikzcd}[column sep=small]
\{p\} \arrow[drr] & \{r\} \arrow[dd] & & \{\neg p, \neg r\} \arrow[dl] \\
& & \{\neg r\} \arrow[dl] & \\
& \Box & &
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=small]
\{\neg q\} \arrow[ddr] & & \{q, \neg p\} \arrow[dr] & & \{p\} \arrow[dl] \\
& & & \{q\} \arrow[dll] & \\
& \Box & & &
\end{tikzcd}$$

Inductive construction of resolution proofs

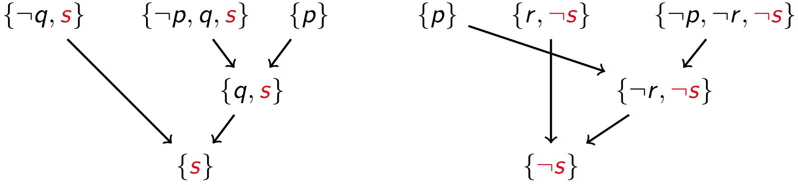
- Reintroduce s and $\neg s$.

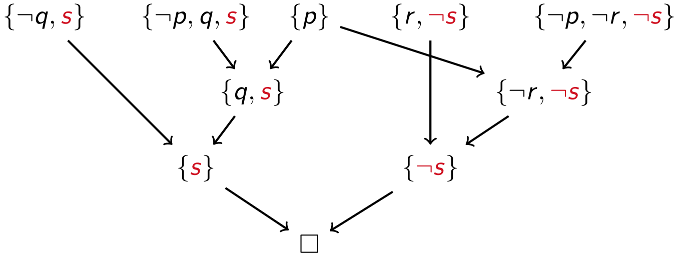
$$F = \{ \{\neg q, s\}, \{\neg p, q, s\}, \{p\}, \{r, \neg s\}, \{\neg p, \neg r, \neg s\} \}$$



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 $F \vdash_{Res} \Box$.

Proof By induction on the number n of distinct atoms in F.

Basis: If $n = 0$ then $F = \{\}$ (but F is unsat.) or $F = \{\Box\}$.

Step:
IH: For every unsat. set of clauses F with n dist. atoms, $F \vdash_{Res} \Box$.
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 $\neg A$.
$F[\bot/A] \equiv$ take F, remove all clauses with $\neg A$, remove all A.

Completeness proof

By IH: there are res. proofs $C_0, \dots, C_m = \Box$ from $F[\bot/A]$ and $D_0, \dots, D_n = \Box$ 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 $C'_m = \Box$ (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:

- either $D'_n = \{\neg A\}$
- or $D'_n = \Box$ (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 $F \vdash_{Res} \Box$.

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 $F \cup \{\neg C\} \vdash_{Res} \Box$

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 $\Box$ is in the final F

Proof F_{init} is unsat. iff $F_{init} \vdash_{Res} \Box$ iff $\Box \in F_{final}$ because the algorithm enumerates all R such that $F_{init} \vdash_{Res} R$.

The algorithm is a decision procedure for unsat. 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

Given formula F and partial assignment $\mathcal{A}$:
$F|_{\mathcal{A}}$ denotes the result of deleting any clause containing a true literal, and deleting all false literals from each remaining clause.

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 $\mathcal{A} \mapsto \mathcal{A}[\top/L]$
3. If $F|_{\mathcal{A}} = \emptyset$, stop and output $\mathcal{A}$.
4. If $F|_{\mathcal{A}} \ni \square$, add new clause C to F by learning procedure.
   If $C = \square$, stop and output UNSAT; otherwise backtrack to highest level where C is unit clause.
   Go to line 2.
5. Apply decision strategy to update $\mathcal{A}$.
   Go to line 2.

Terminology

- State of algorithm is pair $(F, \mathcal{A})$, where F is CNF formula and $\mathcal{A}$ is partial assignment.
  Successful state when $\mathcal{A} \models F$. Conflict state when $\mathcal{A} \not\models F$.

  (Note: conflict state if $F|_{\mathcal{A}} \ni \Box$, successful state if $F|_{\mathcal{A}} = \emptyset$)
- Each assignment $A_i \mapsto b_i$ classifies as decision assignment or implied assignment.
- $A_i \mapsto b_i$ denotes decision assignment with decision variable A_i.
- $A_i \overset{C}{\mapsto} b_i$ denotes an implied assignment arising through unit propagation on clause C.
- Decision level of assignment $A_i \mapsto b_i$ in a given state $(F, \mathcal{A})$ is number of decision assignments in $\mathcal{A}$ that precede $A_i \mapsto b_i$.

Example: start with set of clauses $F = \{C_1, \dots, C_5\}$, where

$$\begin{aligned}
C_1 &= \{\neg A_1, \neg A_4, A_5\}\\
C_2 &= \{\neg A_1, A_6, \neg A_5\}\\
C_3 &= \{\neg A_1, \neg A_6, A_7\}\\
C_4 &= \{\neg A_1, \neg A_7, \neg A_5\}\\
C_5 &= \{A_1, A_4, A_6\}
\end{aligned}$$

Say current assignment is $(A_1 \mapsto 1, A_2 \mapsto 0, A_3 \mapsto 0, A_4 \mapsto 1)$.
Notice $F|_{\mathcal{A}}$ contains unit clause $\{A_5\}$.
Unit propagation further generates $(A_5 \overset{C_1}{\mapsto} 1, A_6 \overset{C_2}{\mapsto} 1, A_7 \overset{C_3}{\mapsto} 1)$.
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)

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 of C is made false by $\mathcal{A}$
3. C mentions only decision variables in $\mathcal{A}$

Clause learning using resolution

Suppose $\mathcal{A} = (A_1 \mapsto b_1, \ldots, A_k \mapsto b_k)$ leads to conflict.
Find associated clauses $D_1, \ldots, D_{k+1}$ by backward induction:

1. $D_{k+1} :=$ any conflict clause of F under $\mathcal{A}$.
2. If $A_i \mapsto b_i$ is decision assignment or A_i not mentioned in D_{i+1}, set $D_i := D_{i+1}$.
3. If $A_i \overset{C_i}{\mapsto} b_i$ is implied assignment and A_i mentioned in D_{i+1}, define D_i to be resolvent of D_{i+1} and C_i with respect to A_i.

$C := A_1$, that is, the final clause A_1 is the learned clause .

Clause learning: example

Conflict of example above:

$C_1 = \{\neg A_1, \neg A_4, A_5\}$	$D_8 := \{\neg A_1, \neg A_7, \neg A_5\}$	(clause C_4)
$C_2 = \{\neg A_1, A_6, \neg A_5\}$	$D_7 := \{\neg A_1, \neg A_5, \neg A_6\}$	(resolve D_8, C_3)
$C_3 = \{\neg A_1, \neg A_6, A_7\}$	$D_6 := \{\neg A_1, \neg A_5\}$	(resolve D_7, C_2)
$C_4 = \{\neg A_1, \neg A_7, \neg A_5\}$	$D_5 := \{\neg A_1, \neg A_4\}$	(resolve D_6, C_1)
$C_5 = \{A_1, A_4, A_6\}$	$D_4 := \{\neg A_1, \neg A_4\}$	
$A_1 \mapsto 1, A_2 \mapsto 0,$	$D_3 := \{\neg A_1, \neg A_4\}$	
$A_3 \mapsto 0, A_4 \mapsto 1,$	$D_2 := \{\neg A_1, \neg A_4\}$	
$A_5 \overset{C_1}{\mapsto} 1, A_6 \overset{C_2}{\mapsto} 1,$	$D_1 := \{\neg A_1, \neg A_4\}$	
$A_7 \overset{C_3}{\mapsto} 1$		

Learned clause D_1 is conflict clause with only decision variables, including top-level one A_1.

Clause learning: example

Intuitively:

- D_1 records that conflict due to decision to make A_1, A_4 true.
- Adding D_1 ensures search does not explore assignments with $A_1 \mapsto 1, A_4 \mapsto 1$.
- DPLL backtracks to highest level where D_1 is unit clause (after $A_1 \mapsto 1$), unit propagation leads to $A_4 \mapsto 0$.

Clause learning

Proposition: The clause learning procedure satisfies the three desiderata.

Proof sketch: Observation: If $A_i \overset{C_i}{\mapsto} b_i$, then the only literal of C_i true under $\mathcal{A}$ is the literal for A_i (that is, C_i contains either A_i or $\neg A_i$, 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 $D_{k+1}, D_k, \cdots D_1 = C$ are conflict clauses.
D_{k+1} is conflict clause by definition.
If D_{i+1} is conflict clause and $D_i = D_{i+1}$, then so is D_i.
If D_{i+1} is conflict clause and $D_i \neq D_{i+1}$, then D_i is the result of resolving D_{i+1} and C_i. By the observation, all literals of D_i are made false by $\mathcal{A}$.

3. C mentions only decision variables in $\mathcal{A}$.
Because every other variable, say A_i, disappears after resolving with D_{i+1} w.r.t. A_i.
Indeed, since $\mathcal{A}$ makes D_{i+1} false, by the observation A_i has opposite signs in D_{i+1} and C_i.

Example (without PLR)

$$\{\neg A_1\}\{A_1, A_3, A_4\}\{\neg A_2, \neg A_5\}\{A_3, \neg A_4, A_5, \neg A_6\}\{A_1, \neg A_2, \neg A_4, A_6\}$$

OLR:	$A_1 \mapsto 0$	$\{A_3, A_4\}$	$\{\neg A_2, \neg A_5\}$	$\{A_3, \neg A_4, A_5, \neg A_6\}$	$\{\neg A_2, \neg A_4, A_6\}$
DE:	$A_2 \mapsto 1$	$\{A_3, A_4\}$	$\{\neg A_5\}$	$\{A_3, \neg A_4, A_5, \neg A_6\}$	$\{\neg A_4, A_6\}$
OLR:	$A_5 \mapsto 0$	$\{A_3, A_4\}$		$\{A_3, \neg A_4, \neg A_6\}$	$\{\neg A_4, A_6\}$
DE:	$A_3 \mapsto 0$	$\{A_4\}$		$\{\neg A_4, \neg A_6\}$	$\{\neg A_4, A_6\}$
OLR:	$A_4 \mapsto 1$			$\{\neg A_6\}$	$\{A_6\}$
OLR:	$A_6 \mapsto 1$			$\{\ \}$	

$D_7 := \{A_3, \neg A_4, A_5, \neg A_6\}$ (conflict clause)

$D_6 := \{A_1, \neg A_2, A_3, \neg A_4, A_5\}$ (resolve D_7, $\{A_1, \neg A_2, \neg A_4, A_6\}$)

$D_5 := \{A_1, \neg A_2, A_3, A_5\}$ (resolve D_6, $\{A_1, A_3, A_4\}$)

$D_4 := \{A_1, \neg A_2, A_3, A_5\}$

$D_3 := \{A_1, \neg A_2, A_3\}$ (resolve D_4, $\{\neg A_2, \neg A_5\}$)

$D_2 := \{A_1, \neg A_2, A_3\}$

$D_1 := \{\neg A_2, A_3\}$ (resolve D_2, $\{\neg A_1\}$)

Backtracking to $\{A_1 \mapsto 0, A_2 \mapsto 1\}$. Unit propagation: $A_3 \mapsto 1$.