Propositional logic

Compiled and modified from slides by
Michael Reniers, Julia Lawall and Neil Jones

Contents

- Syntax of propositional logic
- Semantics of propositional logic
- Semantic entailment
  - Natural deduction proof system
  - Soundness and completeness
- Validity
  - Conjunctive normal forms
- Satisfiability

Syntax of propositional logic

$$F ::= (P) \mid (\neg F) \mid (F \vee F) \mid (F \wedge F) \mid (F \rightarrow F)$$
$$P ::= p \mid q \mid r \mid \ldots$$

- propositional atoms: $p, q, r, \ldots$ for describing declarative sentences such as:
  - All students have to follow the course Programming and Modal Logic
  - 1037 is a prime number
- connectives:

Connective	Symbol	Alternative symbols
negation	$\neg$	$\sim$
disjunction	$\vee$	$\mid$
conjunction	$\wedge$	$\&$
implication	$\rightarrow$	$\Rightarrow$, $\supset$, $\supseteq$

Sometimes also bi-implication ($\leftrightarrow$, $\Leftrightarrow$, $\equiv$) is considered as a connective.

Syntax of propositional logic

Binding priorities

$$\neg$$

$$\vee \qquad \wedge$$

$$\rightarrow \qquad (\leftrightarrow)$$

for reducing the number of brackets.

Also outermost brackets are often omitted.

Semantics of propositional logic

The meaning of a formula depends on:

- The meaning of the propositional atoms (that occur in that formula)
- The meaning of the connectives (that occur in that formula)

Semantics of propositional logic

The meaning of a formula depends on:

- The meaning of the propositional atoms (that occur in that formula)
  - a declarative sentence is either true or false
  - captured as an assignment of truth values ($\mathbb{B} = \{\mathtt{T}, \mathtt{F}\}$) to the propositional atoms: a *valuation* $v : P \longrightarrow \mathbb{B}$
- The meaning of the connectives (that occur in that formula)
  - the meaning of an n-ary connective $\oplus$ is captured by a function $f_{\oplus} : \mathbb{B}^n \longrightarrow \mathbb{B}$
  - usually such functions are specified by means of a truth table.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \rightarrow B$
T	T	F	T	T	T
T	F	F	F	T	F
F	T	T	F	T	T
F	F	T	F	F	T

Exercise

Find the meaning of the formula $(p \to q) \land (q \to r) \to (p \to r)$ by constructing a truth table from the subformulas.

Exercise

Find the meaning of the formula $(p \rightarrow q) \wedge (q \rightarrow r) \rightarrow (p \rightarrow r)$ by constructing a truth table from the subformulas.

p	q	r	$p \rightarrow q$	$q \rightarrow r$	$(p \rightarrow q) \wedge (q \rightarrow r)$	$p \rightarrow r$	$(p \rightarrow q) \wedge (q \rightarrow r) \rightarrow (p \rightarrow r)$
T	T	T	T	T	T	T	T
T	T	F	T	F	F	F	T
T	F	T	F	T	F	T	T
T	F	F	F	T	F	F	T
F	T	T	T	T	T	T	T
F	T	F	T	F	F	T	T
F	F	T	T	T	T	T	T
F	F	F	T	T	T	T	T

Exercise

Find the meaning of the formula $(p \to q) \wedge (q \to r) \to (p \to r)$ by constructing a truth table from the subformulas.

p	q	r	$p \to q$	$q \to r$	$(p \to q) \wedge (q \to r)$	$p \to r$	$(p \to q) \wedge (q \to r) \to (p \to r)$
T	T	T	T	T	T	T	T
T	T	F	T	F	F	F	T
T	F	T	F	T	F	T	T
T	F	F	F	T	F	F	T
F	T	T	T	T	T	T	T
F	T	F	T	F	F	T	T
F	F	T	T	T	T	T	T
F	F	F	T	T	T	T	T

Formally (this is not in the book)

$$[\![\_]\!] : F \to ((P \to \mathbb{B}) \to \mathbb{B})$$

$$
\begin{aligned}
[\![p]\!](v) &= v(p) \\
[\![\neg\phi]\!](v) &= f_\neg([\![\phi]\!](v))
\end{aligned}
\qquad
\begin{aligned}
[\![\phi \wedge \psi]\!](v) &= f_\wedge([\![\phi]\!](v), [\![\psi]\!](v)) \\
[\![\phi \vee \psi]\!](v) &= f_\vee([\![\phi]\!](v), [\![\psi]\!](v)) \\
[\![\phi \to \psi]\!](v) &= f_\to([\![\phi]\!](v), [\![\psi]\!](v))
\end{aligned}
$$

Questions

Our interest lies with the following questions:

- Semantic entailment

  Many logical arguments are of the form: from the assumptions $\phi_1, \cdots, \phi_n$, we know ψ. This is formalised by the *semantic entailment* relation $\models$.

  Formally, $\phi_1, \cdots, \phi_n \models \psi$ iff for all valuations v such that $[\![\phi_i]\!](v) = \mathtt{T}$ for all $1 \leq i \leq n$ we have $[\![\psi]\!](v) = \mathtt{T}$.

- Validity: A formula ϕ is *valid* if $\models \phi$ holds.

- Satisfiability: A formula ϕ is *satisfiable* if there exists a valuation v such that $[\![\phi]\!](v) = \mathtt{T}$.

Semantic entailment

How to establish semantic entailment $\phi_1, \cdots, \phi_n \models \psi$?

Option 1: Construct a truth table.

If the formulas contain m different propositional atoms, the truth table contains 2^m lines!

Option 2: Give a proof.

Suppose that $(p \rightarrow q) \wedge (q \rightarrow r)$. Suppose that p. Then, as $p \rightarrow q$ follows from $(p \rightarrow q) \wedge (q \rightarrow r)$, we have q. Finally, as $q \rightarrow r$ follows from $(p \rightarrow q) \wedge (q \rightarrow r)$, we have r. Thus the formula holds.

Semantic entailment

Proof rules for inferring a conclusion ψ from a list of premises $\phi_1, \cdots, \phi_n$:

$$\phi_1, \cdots, \phi_n \vdash \psi \qquad \text{(sequent)}$$

What is a proof of a sequent $\phi_1, \cdots, \phi_n \vdash \psi$ according to the book (informal definition)?

- Proof rules may be instantiated, i.e. consistent replacement of variables by formulas
- Constructing the proof is filling the gap between the premises and the conclusion by applying a suitable sequence of proof rules.

Natural deduction

Proof rules for **conjunction**:

- $\wedge$ introduction

$$\frac{\phi \qquad \psi}{\phi \wedge \psi} \wedge \mathsf{i}$$

- $\wedge$ elimination

$$\frac{\phi \wedge \psi}{\phi} \wedge \mathsf{e}_1 \qquad \frac{\phi \wedge \psi}{\psi} \wedge \mathsf{e}_2$$

Exercise

Exercise 1.2.1: Prove $(p \wedge q) \wedge r, s \wedge t \vdash q \wedge s$.

Exercise

Exercise 1.2.1: Prove $(p \wedge q) \wedge r, s \wedge t \vdash q \wedge s$.

Linear representation:

1	$(p \wedge q) \wedge r$	premise
2	$s \wedge t$	premise
3	$p \wedge q$	$\wedge e_1$ 1
4	q	$\wedge e_2$ 3
5	s	$\wedge e_1$ 2
6	$q \wedge s$	$\wedge i$ 4, 5

Exercise

Exercise 1.2.1: Prove $(p \wedge q) \wedge r, s \wedge t \vdash q \wedge s$.

Linear representation:

1	$(p \wedge q) \wedge r$	premise
2	$s \wedge t$	premise
3	$p \wedge q$	$\wedge e_1$ 1
4	q	$\wedge e_2$ 3
5	s	$\wedge e_1$ 2
6	$q \wedge s$	$\wedge i$ 4, 5

Tree representation:

$$\dfrac{\dfrac{\dfrac{(p \wedge q) \wedge r}{p \wedge q}\wedge e_1}{q}\wedge e_2 \qquad \dfrac{s \wedge t}{s}\wedge e_1}{q \wedge s}\wedge i$$

Natural deduction

Proof rules for **disjunction**:

- $\vee$ introduction

$$\frac{\phi}{\phi \vee \psi} \vee \mathsf{i}_1 \qquad \frac{\psi}{\phi \vee \psi} \vee \mathsf{i}_2$$

- $\vee$ elimination

$$\frac{\phi \vee \psi \quad \boxed{\begin{array}{c}\phi \\ \vdots \\ \chi\end{array}} \quad \boxed{\begin{array}{c}\psi \\ \vdots \\ \chi\end{array}}}{\chi} \vee \mathsf{e}$$

Exercise

Exercise 1.4.2.(q). Prove

$$(p \wedge q) \vee (p \wedge r) \vdash p \wedge (q \vee r)$$

Exercise

Exercise 1.4.2.(q). Prove

$$(p \wedge q) \vee (p \wedge r) \vdash p \wedge (q \vee r)$$

1	$(p \wedge q) \vee (p \wedge r)$	premise
2	$p \wedge q$	assumption
3	p	$\wedge e_1$ 2
4	q	$\wedge e_2$ 2
5	$q \vee r$	$\vee i_1$ 4
6	$p \wedge (q \vee r)$	$\wedge i$ 3,5
7	$p \wedge r$	assumption
8	p	$\wedge e_1$ 7
9	r	$\wedge e_2$ 7
10	$q \vee r$	$\vee i_2$ 9
11	$p \wedge (q \vee r)$	$\wedge i$ 8,10
12	$p \wedge (q \vee r)$	$\vee e$ 1,2-6,7-11

Natural deduction

Proof rules for **implication**:

- $\longrightarrow$ introduction

$$\frac{\boxed{\begin{array}{c}\phi \\ \vdots \\ \psi\end{array}}}{\phi \longrightarrow \psi} \longrightarrow \mathsf{i}$$

- $\longrightarrow$ elimination

$$\frac{\phi \qquad \phi \longrightarrow \psi}{\psi} \longrightarrow \mathsf{e}$$

Exercise

1. Prove $p \longrightarrow q, q \longrightarrow r \vdash p \longrightarrow r$.

2. Prove $\vdash (p \longrightarrow (q \longrightarrow r)) \longrightarrow (q \longrightarrow (p \longrightarrow r))$.

Exercise

Prove $p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$

Linear representation:

1	$p \rightarrow q$	premise
2	$q \rightarrow r$	premise
3	p	assumption
4	q	$\rightarrow$e 1,3
5	r	$\rightarrow$e 2,4
6	$p \rightarrow r$	$\rightarrow$i 3-5

(Lines 3–5 are enclosed in a box.)

Tree representation (assumption management more difficult):

$$\dfrac{\boxed{\dfrac{\dfrac{p \rightarrow q \qquad p}{q}\ \rightarrow\text{e} \qquad q \rightarrow r}{r}\ \rightarrow\text{e}}}{p \rightarrow r}\ \rightarrow\text{i}$$

Natural deduction

Proof rules for **negation**:

- $\neg$ introduction

$$\frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg\mathsf{i}$$

- $\neg$ elimination

$$\frac{\phi \qquad \neg\phi}{\bot}\ \neg\mathsf{e}$$

Example: $\vdash p \to (\neg p \to q)$

Natural deduction

Proof rules for **falsum**:

- $\bot$ introduction: there are no proof rules for the introduction of $\bot$
- $\bot$ elimination

$$\frac{\bot}{\phi}\ \bot\text{e}$$

Proof rules for **double negation**:

- $\neg\neg$ elimination

$$\frac{\neg\neg\phi}{\phi}\ \neg\neg\text{e}$$

Natural deduction

Derived rules (derivation in book):

- Modus Tollens $$\frac{\phi \to \psi \qquad \neg\psi}{\neg\phi}\text{MT}$$

- $\neg\neg$ introduction $$\frac{\phi}{\neg\neg\phi}\neg\neg\text{i}$$

- Reduction Ad Absurdum / Proof by contradiction $$\frac{\boxed{\begin{array}{c}\neg\phi\\ \vdots\\ \bot\end{array}}}{\phi}\text{RAA}$$

- Law of the Excluded Middle / Tertium Non Datur $$\frac{}{\phi \vee \neg\phi}\text{LEM}$$

Natural deduction

Soundness of natural deduction

if $\phi_1, \cdots, \phi_n \vdash \psi$, then $\phi_1, \cdots, \phi_n \models \psi$

Completeness of natural deduction

if $\phi_1, \cdots, \phi_n \models \psi$, then $\phi_1, \cdots, \phi_n \vdash \psi$

Deciding validity and satisfiability of propositional formulas

- Validity: A formula ϕ is *valid* if for any valuations v, $[\![\phi]\!](v) = \mathtt{T}$.
- Satisfiability: A formula ϕ is *satisfiable* if there exists a valuation v such that $[\![\phi]\!](v) = \mathtt{T}$.

Deciding validity and satisfiability of propositional formulas

- Validity: A formula ϕ is *valid* if for any valuations v, $[\![\phi]\!](v) = \mathtt{T}$.
- Satisfiability: A formula ϕ is *satisfiable* if there exists a valuation v such that $[\![\phi]\!](v) = \mathtt{T}$.

Examples

$p \wedge q$	valid? satisfiable?
$p \rightarrow (q \rightarrow p)$	valid? satisfiable?
$p \wedge \neg p$	valid? satisfiable?

Deciding validity and satisfiability of propositional formulas

- Validity: A formula ϕ is *valid* if for any valuations v, $[\![\phi]\!](v) = \mathtt{T}$.
- Satisfiability: A formula ϕ is *satisfiable* if there exists a valuation v such that $[\![\phi]\!](v) = \mathtt{T}$.

Examples

$p \wedge q$	satisfiable
$p \rightarrow (q \rightarrow p)$	valid
$p \wedge \neg p$	unsatisfiable

Given a propositional formula ϕ, how to check whether it is valid? satisfiable?

Deciding validity

What are the means to decide whether or not a given formula ϕ is valid?

- Use techniques for semantic entailment (e.g., natural deduction).
- Use a calculus for semantical equivalence to prove that $\phi \equiv \top$.
- *Transform ϕ into some normal form that is semantically equivalent and then apply dedicated techniques (syntactic).*

  ϕ and ψ are semantically equivalent (not. $\phi \equiv \psi$) iff $\phi \models \psi$ and $\psi \models \phi$.

A decision procedure for validity can be used for semantic entailment.
Lemma (1.41):

$$\phi_1, \cdots, \phi_n \models \psi \text{ iff } \models \phi_1 \rightarrow (\phi_2 \rightarrow \cdots \rightarrow (\phi_n \rightarrow \psi))$$

Deciding Validity

- If I am wealthy, then I am happy. I am happy. Therefore, I am wealthy.
- If John drinks beer, he is at least 18 years old. John does not drink beer. Therefore, John is not yet 18 years old.
- If girls are blonde, they are popular with boys. Ugly girls are unpopular with boys. Intellectual girls are ugly. Therefore, blonde girls are not intellectual.
- If I study, then I will not fail basket weaving 101. If I do not play cards to often, then I will study. I failed basket weaving 101. Therefore, I played cards too often.

Deciding validity

Conjunctive Normal Forms

A literal is either an atom p or the negation of an atom $\neg p$.

A formule ϕ is in conjunctive normal form **(CNF)** if it is a conjunction of a number of disjunctions of literals only.

$L ::= P \mid \neg P$	literal
$C ::= L \mid C \vee C$	clause
$CNF ::= C \mid CNF \wedge CNF$	CNF

Examples

- p and $\neg p$ are in CNF;
- $\neg\neg p$ is not in CNF;
- $p \wedge \neg p$ and $(p \vee \neg r) \wedge (\neg r \vee s) \wedge q$ are in CNF;
- $(p \wedge \neg q) \vee q$ is not in CNF.

Deciding validity

Usefulness of CNF

- Deciding validity of formulas in CNF is easy!

  $$\boxed{C_1 \wedge C_2 \wedge \cdots \wedge C_n} \tag{CNF}$$

  Each clause has to be valid.

  $$\boxed{L_1 \vee L_2 \vee \cdots \vee L_m} \tag{C}$$

  Lemma (1.43): $\models L_1 \vee \cdots \vee L_m$ iff there are i and j $(1 \leq i, j \leq m)$ such that L_i and $\neg L_j$ are syntactically equal.

- Any formula be transformed into an equivalent formula in CNF!

Deciding validity

Transformation into CNF

1. **Remove all occurrences of $\longrightarrow$.**

   Done by the algorithm `IF`

   Input: formula
   Output: formula without $\longrightarrow$

2. **Obtain a 'negation normal form' (only atoms are negated!).**

$$N ::= P \mid \neg P \mid (N \vee N) \mid (N \wedge N)$$
$$P ::= p \mid q \mid r \mid \cdots$$

   Done by the algorithm `NNF`

   Input: formula without $\longrightarrow$
   Output: formula in NNF

3. **Apply distribution laws**

   Done by the algorithm `CNF`

   Input: formula in NNF
   Output: formula in CNF

Therefore, $\texttt{CNF}(\texttt{NNF}(\texttt{IF}(\phi)))$ is in CNF and semantically equivalent with ϕ.

Deciding validity

Transformation into CNF. The algorithm `IF`

Idea: Apply the following replacement until it can not be applied anymore: $\phi \rightarrow \psi$ replace by $\neg\phi \vee \psi$

Inductive definition of `IF`:

$$
\begin{array}{lcl}
\mathtt{IF}(p) & = & p \\
\mathtt{IF}(\neg\phi) & = & \neg\mathtt{IF}(\phi) \\
\mathtt{IF}(\phi_1 \wedge \phi_2) & = & \mathtt{IF}(\phi_1) \wedge \mathtt{IF}(\phi_2) \\
\mathtt{IF}(\phi_1 \vee \phi_2) & = & \mathtt{IF}(\phi_1) \vee \mathtt{IF}(\phi_2) \\
\mathtt{IF}(\phi_1 \rightarrow \phi_2) & = & \neg\mathtt{IF}(\phi_1) \vee \mathtt{IF}(\phi_2)
\end{array}
$$

Properties of `IF`:

- `IF` is well-defined (terminates for any input)
- $\mathtt{IF}(\phi) \equiv \phi$ (the output of `IF` and the input of `IF` are semantically equivalent)
- $\mathtt{IF}(\phi)$ is an implication-free formula for any formula ϕ

Deciding validity

Transformation into CNF. The algorithm $\mathtt{NNF}$

Idea: apply the following replacements until none can be applied anymore:

$\neg\neg\phi$	replace by	ϕ
$\neg(\phi \wedge \psi)$	replace by	$\neg\phi \vee \neg\psi$
$\neg(\phi \vee \psi)$	replace by	$\neg\phi \wedge \neg\psi$

Inductive definition of $\mathtt{NNF}$:

$$
\begin{array}{lcl}
\mathtt{NNF}(p) & = & p \\
\mathtt{NNF}(\neg p) & = & \neg p \\
\mathtt{NNF}(\neg\neg\phi) & = & \mathtt{NNF}(\phi) \\
\mathtt{NNF}(\neg(\phi_1 \wedge \phi_2)) & = & \mathtt{NNF}(\neg\phi_1 \vee \neg\phi_2) \\
\mathtt{NNF}(\neg(\phi_1 \vee \phi_2)) & = & \mathtt{NNF}(\neg\phi_1 \wedge \neg\phi_2) \\
\mathtt{NNF}(\phi_1 \wedge \phi_2) & = & \mathtt{NNF}(\phi_1) \wedge \mathtt{NNF}(\phi_2) \\
\mathtt{NNF}(\phi_1 \vee \phi_2) & = & \mathtt{NNF}(\phi_1) \vee \mathtt{NNF}(\phi_2)
\end{array}
$$

Properties of $\mathtt{NNF}$:

- $\mathtt{NNF}$ is well-defined (terminates for any input)
- $\mathtt{NNF}(\phi) \equiv \phi$ (the output of $\mathtt{NNF}$ and the input of $\mathtt{NNF}$ are semantically equivalent)
- $\mathtt{NNF}(\phi)$ is a NNF for any implication-free formula ϕ

Deciding validity

Transformation into CNF. The algorithm $\texttt{CNF}$

Idea: apply until no longer possible:

$$(\phi_1 \land \phi_2) \lor \psi \quad \text{replace by} \quad (\phi_1 \lor \psi) \land (\phi_2 \lor \psi)$$
$$\phi \lor (\psi_1 \land \psi_2) \quad \text{replace by} \quad (\phi \lor \psi_1) \land (\phi \lor \psi_2)$$

Inductive definition of $\texttt{CNF}$:

$$\begin{array}{lcl} \texttt{CNF}(p) & = & p \\ \texttt{CNF}(\neg p) & = & \neg p \\ \texttt{CNF}(\phi_1 \land \phi_2) & = & \texttt{CNF}(\phi_1) \land \texttt{CNF}(\phi_2) \\ \texttt{CNF}(\phi_1 \lor \phi_2) & = & \texttt{D}(\texttt{CNF}(\phi_1), \texttt{CNF}(\phi_2)) \end{array}$$

with

$$\texttt{D}(\phi_1, \phi_2) = \begin{cases} \texttt{D}(\phi_{11}, \phi_2) \land \texttt{D}(\phi_{12}, \phi_2) & \phi_1 = \phi_{11} \land \phi_{12} \\ \texttt{D}(\phi_1, \phi_{21}) \land \texttt{D}(\phi_1, \phi_{22}) & \phi_2 = \phi_{21} \land \phi_{22} \\ \phi_1 \lor \phi_2 & \text{otherwise} \end{cases}$$

Properties of $\texttt{CNF}$ (and $\texttt{D}$):

- $\texttt{CNF}$ and $\texttt{D}$ are well-defined
- $\texttt{D}(\phi, \psi) \equiv \phi \lor \psi$ and $\texttt{CNF}(\phi) \equiv \phi$
- $\texttt{CNF}(\phi)$ is in CNF for any formula ϕ in NNF and $\texttt{D}(\phi, \psi)$ is in CNF for any formulas ϕ and ψ in CNF

Example

Find a CNF for $p \vee \neg q \rightarrow r$.

$$
\begin{aligned}
& p \vee \neg q \rightarrow r \\
\mapsto \quad & \neg(p \vee \neg q) \vee r
\end{aligned}
$$

$$
\begin{aligned}
& \neg(p \vee \neg q) \vee r \\
\mapsto \quad & (\neg p \wedge \neg\neg q) \vee r \\
\mapsto \quad & (\neg p \wedge q) \vee r
\end{aligned}
$$

$$
\begin{aligned}
& (\neg p \wedge q) \vee r \\
\mapsto \quad & (\neg p \vee r) \wedge (q \vee r)
\end{aligned}
$$

Example

Validity of $((p \leftrightarrow q) \leftrightarrow r) \leftrightarrow s$.

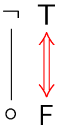
CNF: ??

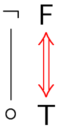
SAT Solver

- Finding satisfying valuations to a propositional formula.

Forcing laws – negation

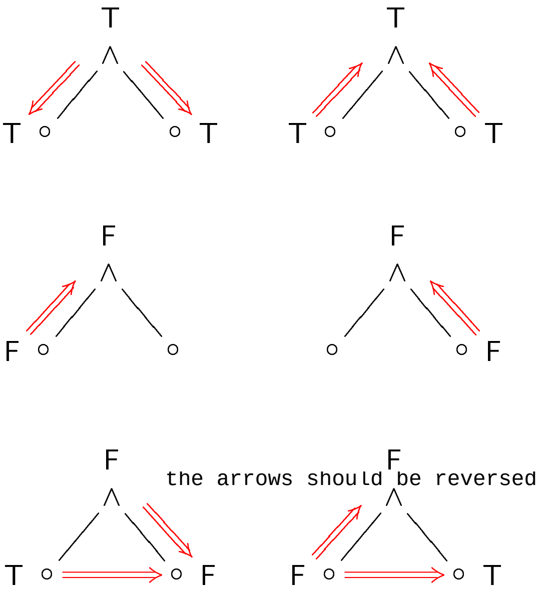
ϕ	$\neg\phi$
T	F
F	T





Forcing laws – conjunction

ϕ	ψ	$\phi \wedge \psi$
T	T	T
T	F	F
F	T	F
F	F	F



Other laws possible,
but $\neg$ and $\wedge$
are adequate

Using the SAT solver

1. Convert to $\neg$ and $\wedge$.

   $$T(p) = p \qquad T(\neg\phi) = \neg T(\phi)$$
   $$T(\phi \wedge \psi) = T(\phi) \wedge T(\psi) \qquad T(\phi \vee \psi) = \neg(\neg T(\phi) \wedge \neg T(\psi))$$
   $$T(\phi \to \psi) = \neg(T(\phi) \wedge \neg T(\psi))$$

   Linear growth in formula size (no distributivity).

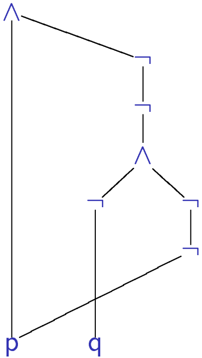
2. Translate the formula to a DAG, sharing common subterms.

3. Set the root to T and apply the forcing rules.

Satisfiable if all nodes are consistently annotated.

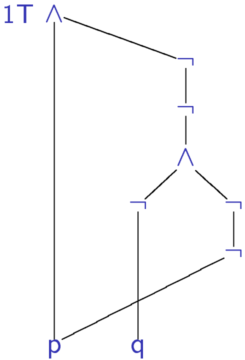
Example: satisfiability

Formula: $p \wedge \neg(q \vee \neg p) \equiv p \wedge \neg\neg(\neg q \wedge \neg\neg p)$



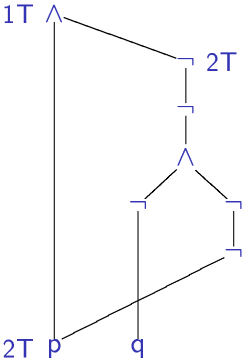
Example: satisfiability

Formula: $p \wedge \neg(q \vee \neg p) \equiv p \wedge \neg\neg(\neg q \wedge \neg\neg p)$



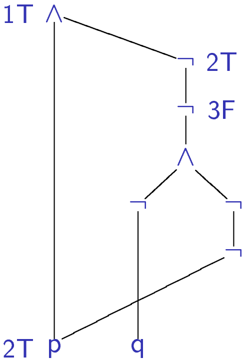
Example: satisfiability

Formula: $p \wedge \neg(q \vee \neg p) \equiv p \wedge \neg\neg(\neg q \wedge \neg\neg p)$



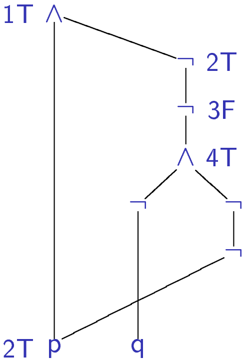
Example: satisfiability

Formula: $p \wedge \neg(q \vee \neg p) \equiv p \wedge \neg\neg(\neg q \wedge \neg\neg p)$



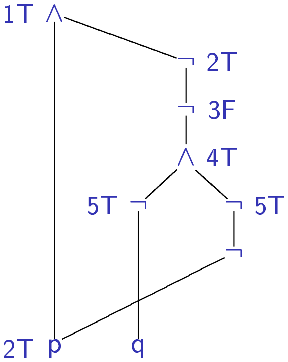
Example: satisfiability

Formula: $p \wedge \neg(q \vee \neg p) \equiv p \wedge \neg\neg(\neg q \wedge \neg\neg p)$



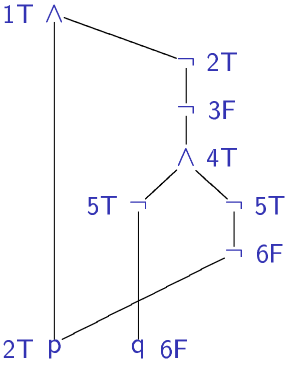
Example: satisfiability

Formula: $p \wedge \neg(q \vee \neg p) \equiv p \wedge \neg\neg(\neg q \wedge \neg\neg p)$



Example: satisfiability

Formula: $p \wedge \neg(q \vee \neg p) \equiv p \wedge \neg\neg(\neg q \wedge \neg\neg p)$



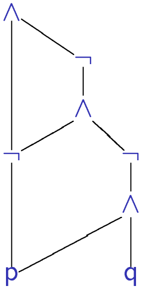
Satisfiable?

Example: validity

Formula: $(p \vee (p \wedge q)) \rightarrow p$

Valid if $\neg((p \vee (p \wedge q)) \rightarrow p)$ is not satisfiable

Translated formula: $\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p$

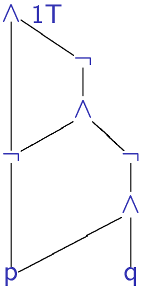


Example: validity

Formula: $(p \vee (p \wedge q)) \to p$

Valid if $\neg((p \vee (p \wedge q)) \to p)$ is not satisfiable

Translated formula: $\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p$

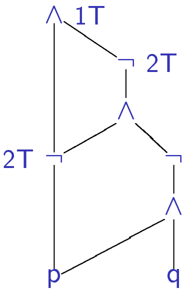


Example: validity

Formula: $(p \vee (p \wedge q)) \rightarrow p$

Valid if $\neg((p \vee (p \wedge q)) \rightarrow p)$ is not satisfiable

Translated formula: $\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p$

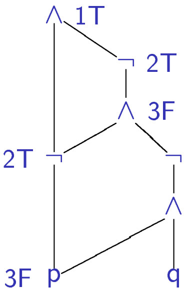


Example: validity

Formula: $(p \vee (p \wedge q)) \rightarrow p$

Valid if $\neg((p \vee (p \wedge q)) \rightarrow p)$ is not satisfiable

Translated formula: $\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p$

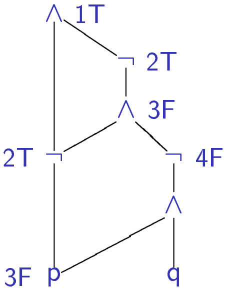


Example: validity

Formula: $(p \vee (p \wedge q)) \to p$

Valid if $\neg((p \vee (p \wedge q)) \to p)$ is not satisfiable

Translated formula: $\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p$

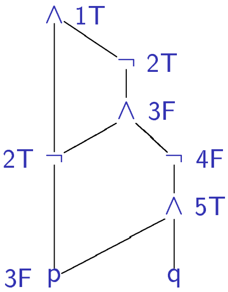


Example: validity

Formula: $(p \vee (p \wedge q)) \rightarrow p$

Valid if $\neg((p \vee (p \wedge q)) \rightarrow p)$ is not satisfiable

Translated formula: $\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p$

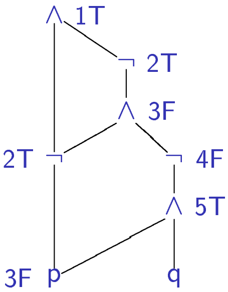


Example: validity

Formula: $(p \vee (p \wedge q)) \rightarrow p$

Valid if $\neg((p \vee (p \wedge q)) \rightarrow p)$ is not satisfiable

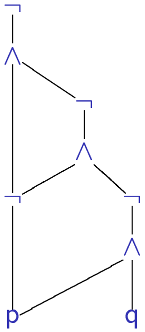
Translated formula: $\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p$



Contradiction.

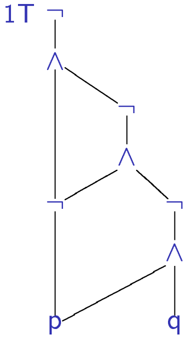
Example: satisfiability

Formula: $(p \vee (p \wedge q)) \rightarrow p \equiv \neg(\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p)$



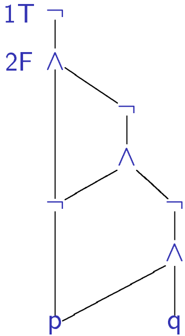
Example: satisfiability

Formula: $(p \vee (p \wedge q)) \rightarrow p \equiv \neg(\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p)$



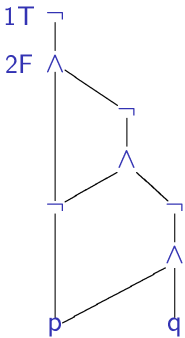
Example: satisfiability

Formula: $(p \vee (p \wedge q)) \rightarrow p \equiv \neg(\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p)$



Example: satisfiability

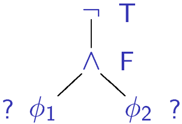
Formula: $(p \vee (p \wedge q)) \rightarrow p \equiv \neg(\neg(\neg p \wedge \neg(p \wedge q)) \wedge \neg p)$



Now what?

Limitation of the SAT solver algorithm

Fails for all formulas of the form $\neg(\phi_1 \wedge \phi_2)$.

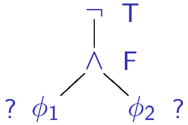


Some are valid, and thus satisfiable:

$$\top$$

Limitation of the SAT solver algorithm

Fails for all formulas of the form $\neg(\phi_1 \wedge \phi_2)$.

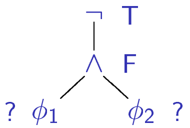


Some are valid, and thus satisfiable:

$$\top \equiv p \to p$$

Limitation of the SAT solver algorithm

Fails for all formulas of the form $\neg(\phi_1 \wedge \phi_2)$.

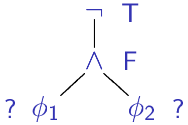


Some are valid, and thus satisfiable:

$$\top \equiv p \rightarrow p \equiv \neg(p \wedge \neg p)$$

Limitation of the SAT solver algorithm

Fails for all formulas of the form $\neg(\phi_1 \wedge \phi_2)$.



Some are valid, and thus satisfiable:

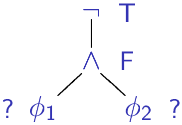
$$\top \equiv p \rightarrow p \equiv \neg(p \wedge \neg p)$$

Some are not valid, and thus not satisfiable:

$\bot$

Limitation of the SAT solver algorithm

Fails for all formulas of the form $\neg(\phi_1 \wedge \phi_2)$.



Some are valid, and thus satisfiable:

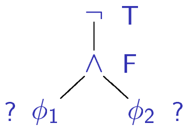
$$\top \equiv p \to p \equiv \neg(p \wedge \neg p)$$

Some are not valid, and thus not satisfiable:

$\bot \equiv \neg\top$

Limitation of the SAT solver algorithm

Fails for all formulas of the form $\neg(\phi_1 \land \phi_2)$.



Some are valid, and thus satisfiable:

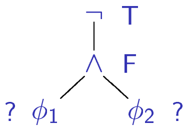
$$\top \equiv p \rightarrow p \equiv \neg(p \land \neg p)$$

Some are not valid, and thus not satisfiable:

$\bot \equiv \neg\top \equiv \neg(\top \land \top)$

Limitation of the SAT solver algorithm

Fails for all formulas of the form $\neg(\phi_1 \wedge \phi_2)$.



Some are valid, and thus satisfiable:

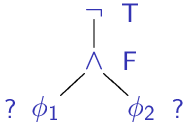
$$\top \equiv p \to p \equiv \neg(p \wedge \neg p)$$

Some are not valid, and thus not satisfiable:

$$\bot \equiv \neg\top \equiv \neg(\top \wedge \top) \equiv \neg(p \to p \wedge p \to p)$$

Limitation of the SAT solver algorithm

Fails for all formulas of the form $\neg(\phi_1 \wedge \phi_2)$.



Some are valid, and thus satisfiable:

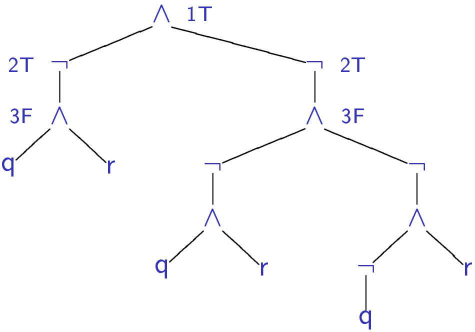
$$\top \equiv p \to p \equiv \neg(p \wedge \neg p)$$

Some are not valid, and thus not satisfiable:

$$\bot \equiv \neg\top \equiv \neg(\top\wedge\top) \equiv \neg(p \to p\wedge p \to p) \equiv \neg(\neg(p\wedge\neg p)\wedge\neg(p\wedge\neg p))$$

Extending the algorithm

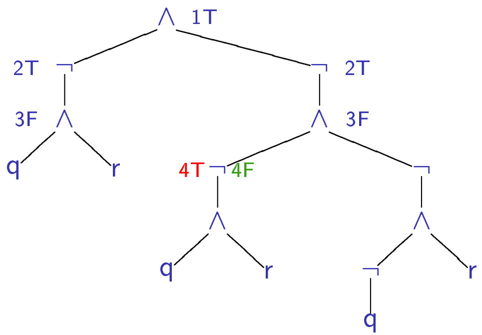
Formula: $\neg(q \land r) \land \neg(\neg(q \land r) \land \neg(\neg q \land r))$



Idea: pick a node and try both possibilities

Extending the algorithm

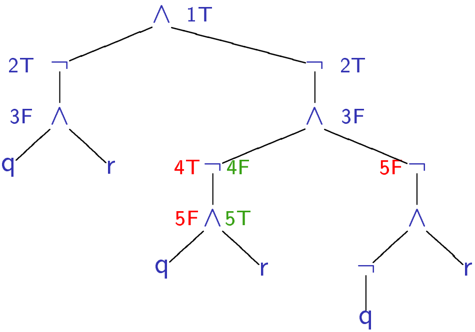
Formula: $\neg(q \wedge r) \wedge \neg(\neg(q \wedge r) \wedge \neg(\neg q \wedge r))$



Idea: pick a node and try both possibilities

Extending the algorithm

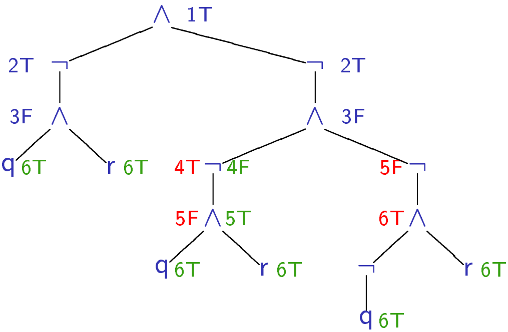
Formula: $\neg(q \wedge r) \wedge \neg(\neg(q \wedge r) \wedge \neg(\neg q \wedge r))$



Idea: pick a node and try both possibilities

Extending the algorithm

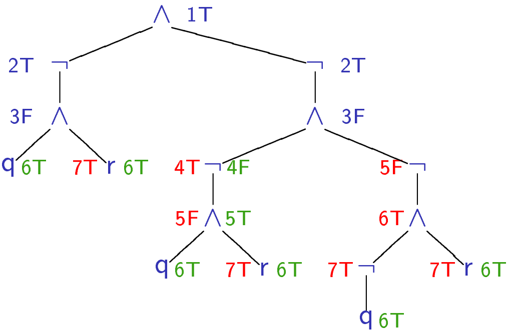
Formula: $\neg(q \land r) \land \neg(\neg(q \land r) \land \neg(\neg q \land r))$



Idea: pick a node and try both possibilities

Extending the algorithm

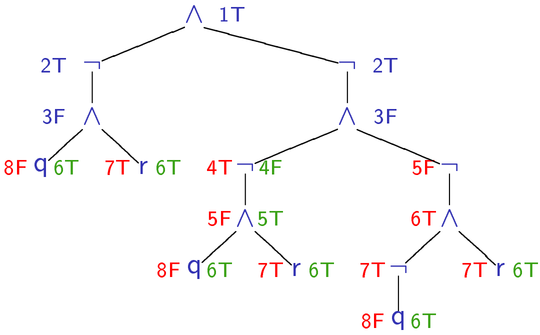
Formula: $\neg(q \wedge r) \wedge \neg(\neg(q \wedge r) \wedge \neg(\neg q \wedge r))$



Idea: pick a node and try both possibilities

Extending the algorithm

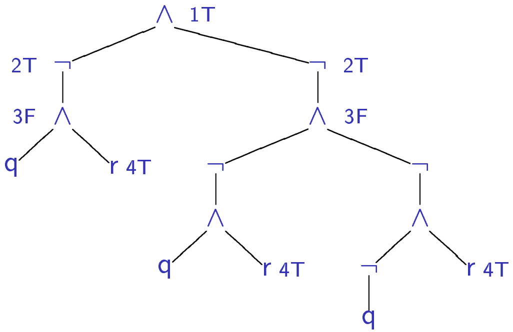
Formula: $\neg(q \wedge r) \wedge \neg(\neg(q \wedge r) \wedge \neg(\neg q \wedge r))$



r is true in both cases

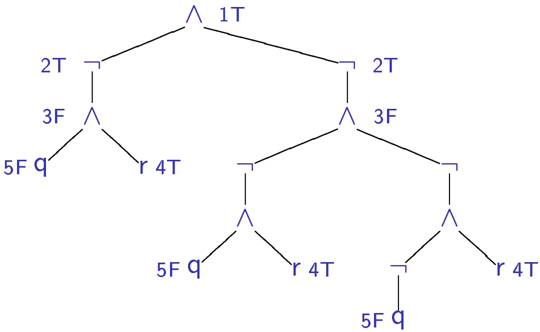
Using the value of r

Formula: $\neg(q \wedge r) \wedge \neg(\neg(q \wedge r) \wedge \neg(\neg q \wedge r))$



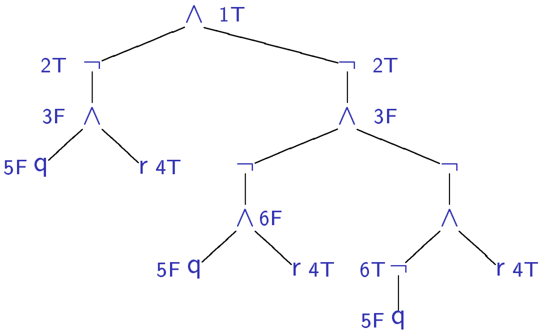
Using the value of r

Formula: $\neg(q \wedge r) \wedge \neg(\neg(q \wedge r) \wedge \neg(\neg q \wedge r))$



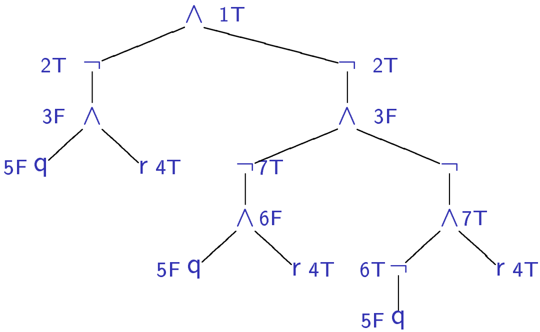
Using the value of r

Formula: $\neg(q \wedge r) \wedge \neg(\neg(q \wedge r) \wedge \neg(\neg q \wedge r))$



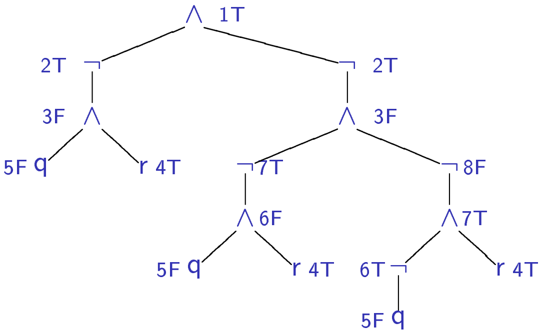
Using the value of r

Formula: $\neg(q \wedge r) \wedge \neg(\neg(q \wedge r) \wedge \neg(\neg q \wedge r))$



Using the value of r

Formula: $\neg(q \wedge r) \wedge \neg(\neg(q \wedge r) \wedge \neg(\neg q \wedge r))$



Satisfiable.

Extended algorithm

Algorithm:

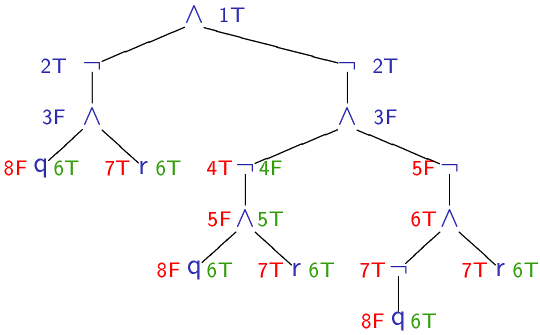
1. Pick an unmarked node and add temporary T and F marks.
2. Use the forcing rules to propagate both marks.
3. If both marks lead to a contradiction, report a contradiction.
4. If both marks lead to some node having the same value, permanently assign the node that value.
5. Erase the remaining temporary marks and continue.

Complexity $O(n^3)$:

1. Testing each unmarked node: $O(n)$
2. Testing a given unmarked node: $O(n)$
3. Repeating the whole thing when a new node is marked: $O(n)$

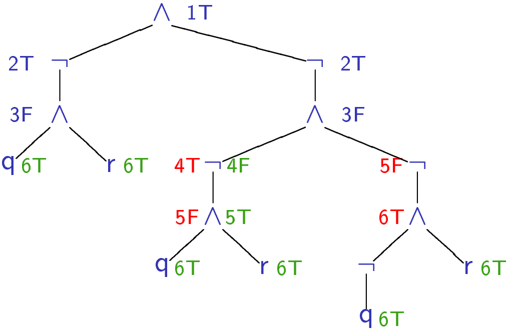
An optimization

Formula: $\neg(q \land r) \land \neg(\neg(q \land r) \land \neg(\neg q \land r))$



We could stop here: red values give a complete and consistent valuation.

Another optimization



- Contradiction in the leftmost subtree.
- No need to analyze q, etc.
- Permanently mark “4T4F” as T.

Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

Perform backtracking search over values of variables

Try to satisfy each clause

M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM, 5:394–397, 1962*

Slides from Aarti Gupta

Basic DLL Search

Performs backtracking search over variable assignments

Basic definitions

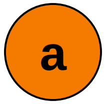
Under a given partial assignment (PA) to variables

- A variable may be
  - **assigned** (true/false literal)
  - **unassigned**.

- A clause may be
  - **satisfied** (≥1 true literal)
  - **unsatisfied** (all false literals)
  - **unit** (one unassigned literal, rest false)
  - **unresolved** (otherwise)

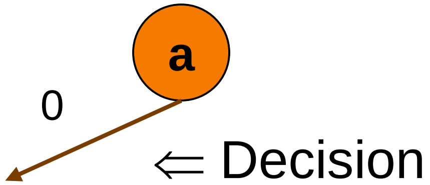
Basic DLL Search

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



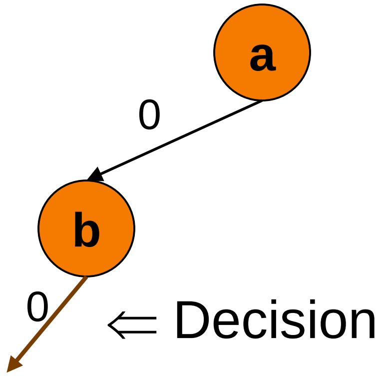
Basic DLL Search

→ (a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
→ (a’ + b + c’)
→ (a’ + b’ + c)



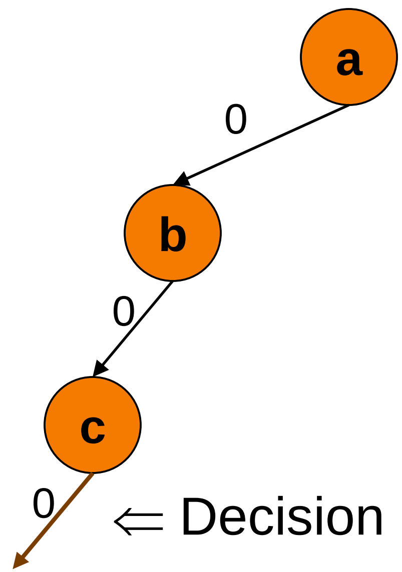
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
→ (b' + c' + d)
(a' + b + c')
(a' + b' + c)



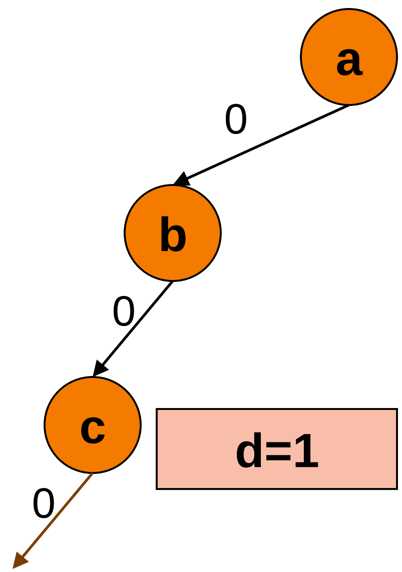
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
→ (a + c' + d)
→ (a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

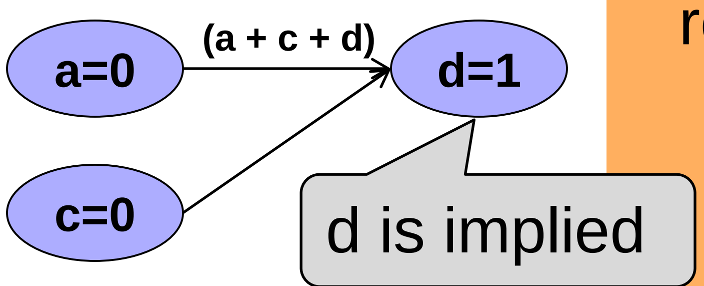


Basic DLL Search

- (a’ + b + c)
- → **(a + c + d)** ⇐ Unit
- (a + c + d’)
- (a + c’ + d)
- (a + c’ + d’)
- (b’ + c’ + d)
- (a’ + b + c’)
- (a’ + b’ + c)



Implication Graph



BCP: Boolean Constraint Propagation repeatedly applies *Unit Clause Rule*

$$\frac{\text{lit} \qquad \text{clause}[\text{lit}']}{\text{clause}[\bot]}$$

Basic DLL Search

(a' + b + c)

→ (a + c + d)

→ (a + c + d') ⇐ Unit

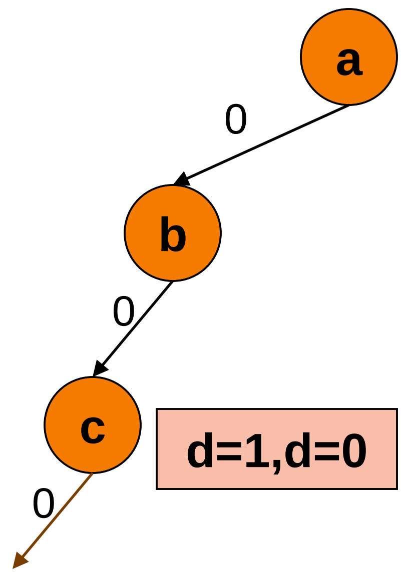
(a + c' + d)

(a + c' + d')

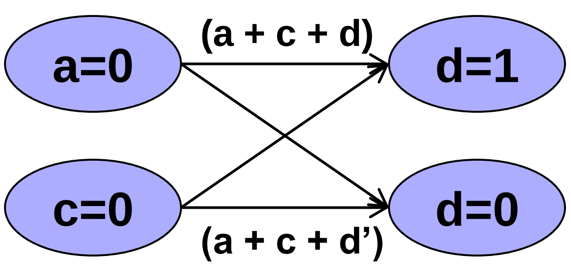
(b' + c' + d)

(a' + b + c')

(a' + b' + c)

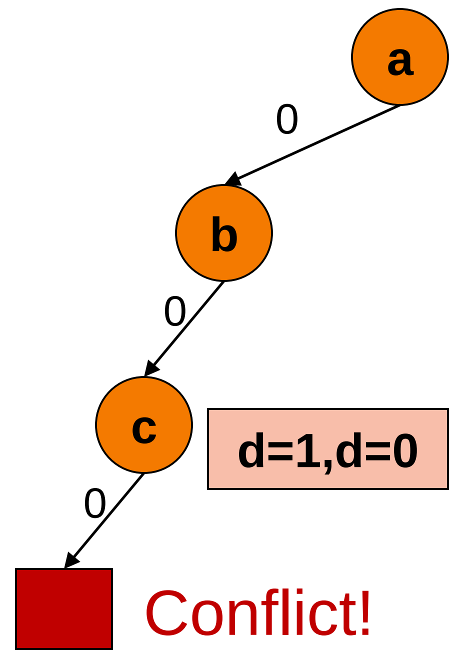


Implication Graph

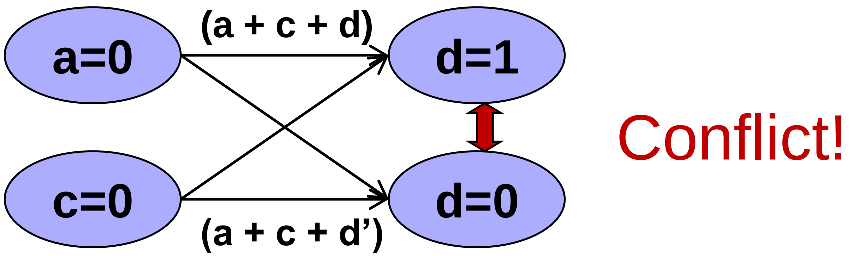


Basic DLL Search

- (a’ + b + c)
- → (a + c + d)
- → (a + c + d’)
- (a + c’ + d)
- (a + c’ + d’)
- (b’ + c’ + d)
- (a’ + b + c’)
- (a’ + b’ + c)

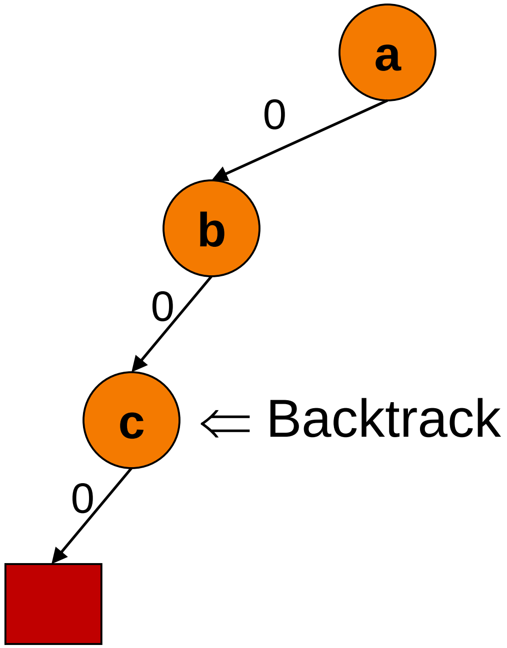


Implication Graph



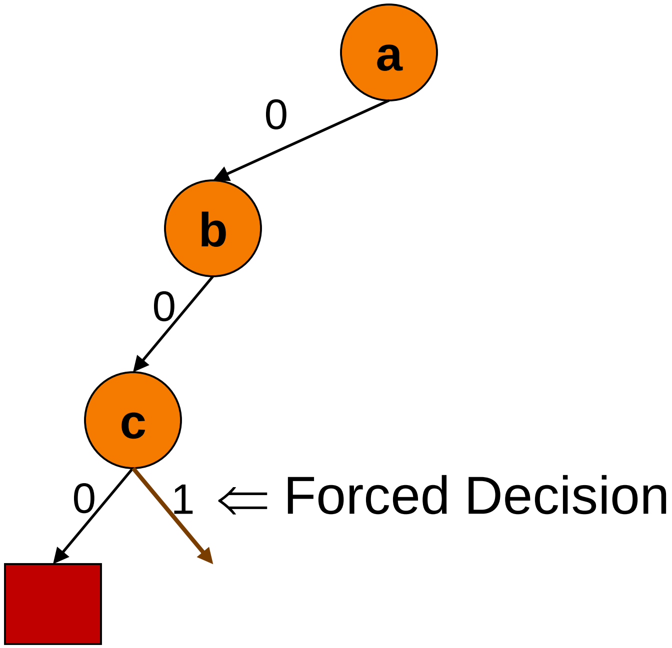
Basic DLL Search

(a' + b + c)
→ (a + c + d)
→ (a + c + d')
→ (a + c' + d)
→ (a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



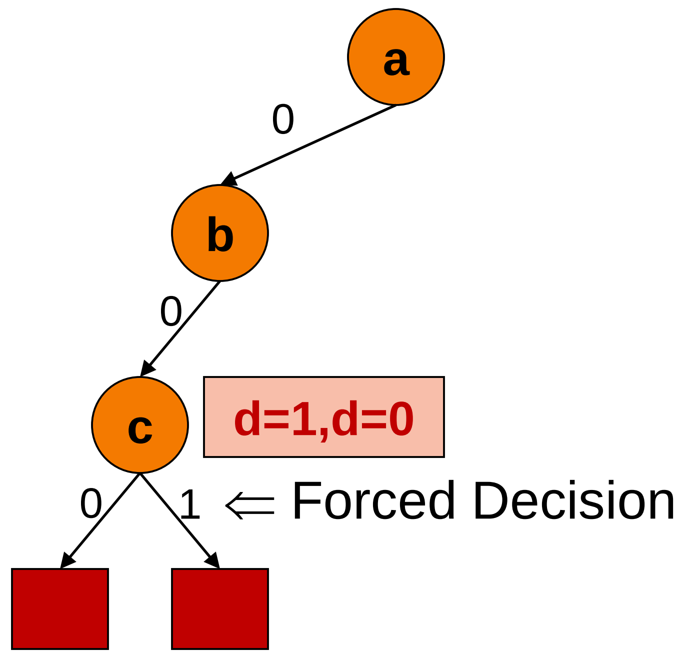
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
→ (a + c' + d)
→ (a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

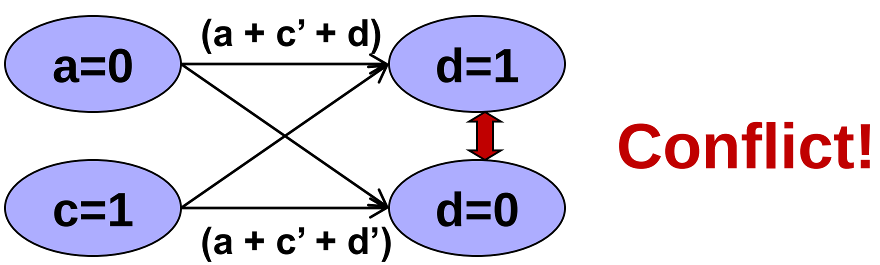


Basic DLL Search

(a’ + b + c)
(a + c + d)
(a + c + d’)
→ (a + c’ + d)
→ (a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)

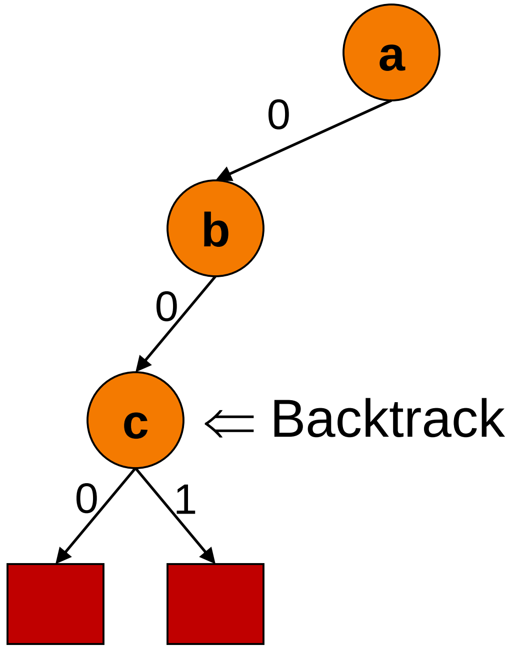


Implication Graph



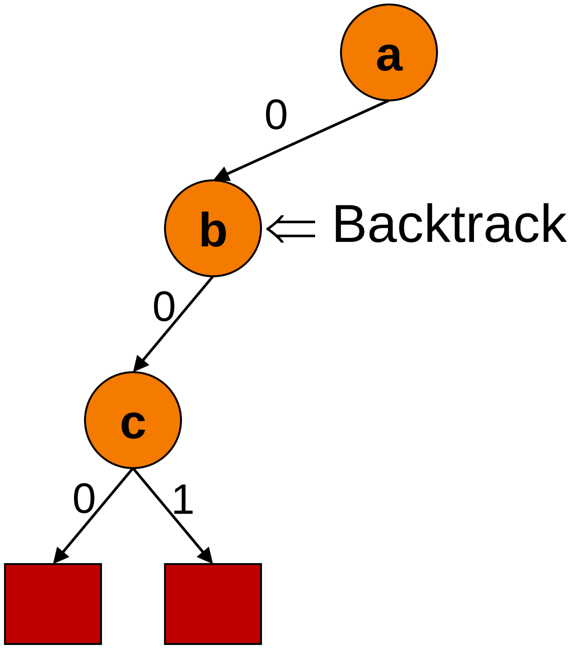
Basic DLL Search

(a’ + b + c)
→ (a + c + d)
→ (a + c + d’)
→ (a + c’ + d)
→ (a + c’ + d’)
→ (b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



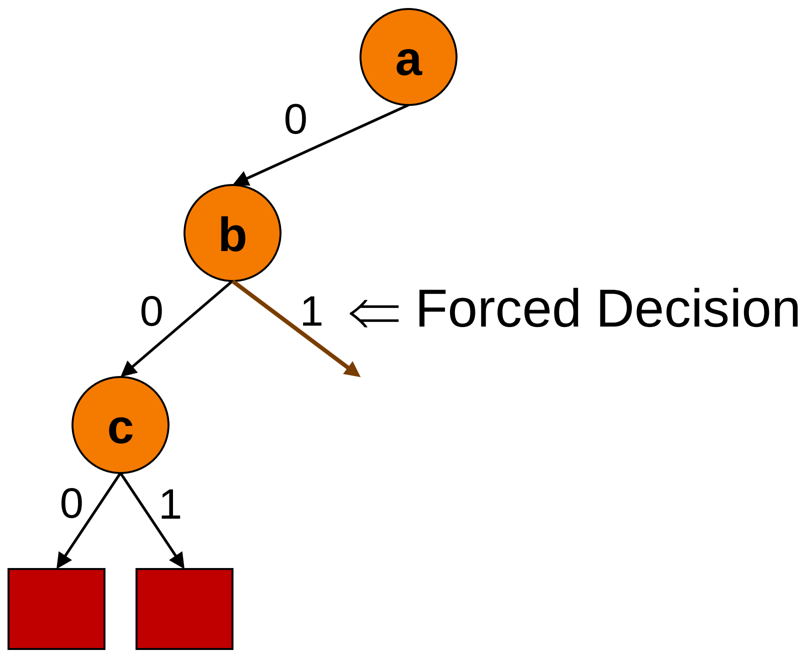
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



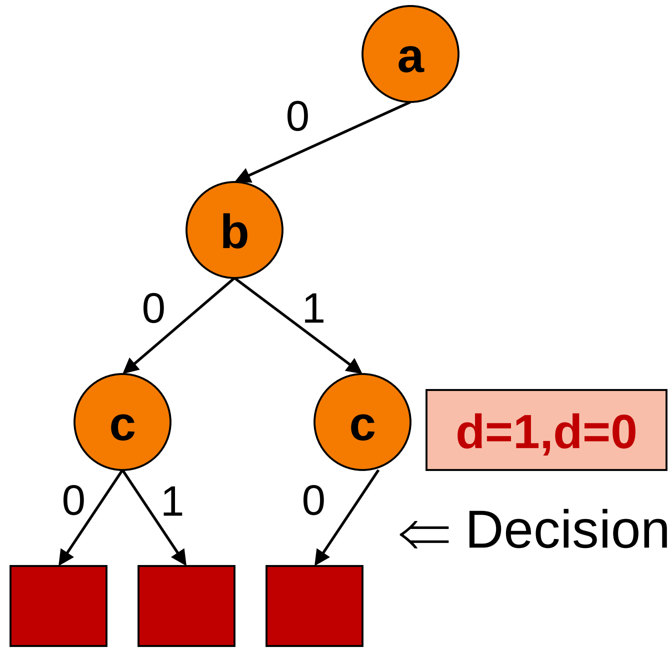
Basic DLL Search

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)

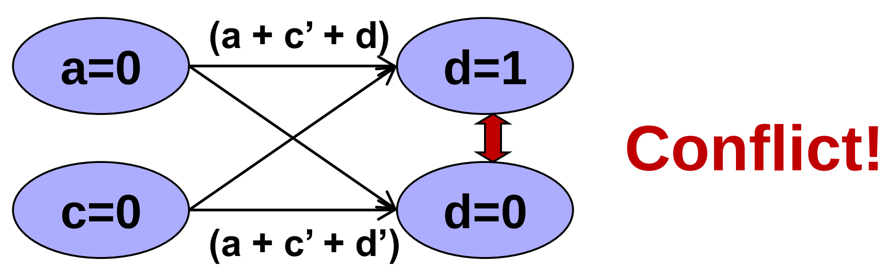


Basic DLL Search

(a’ + b + c)
→ (a + c + d)
→ (a + c + d’)
→ (a + c’ + d)
→ (a + c’ + d’)
→ (b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)

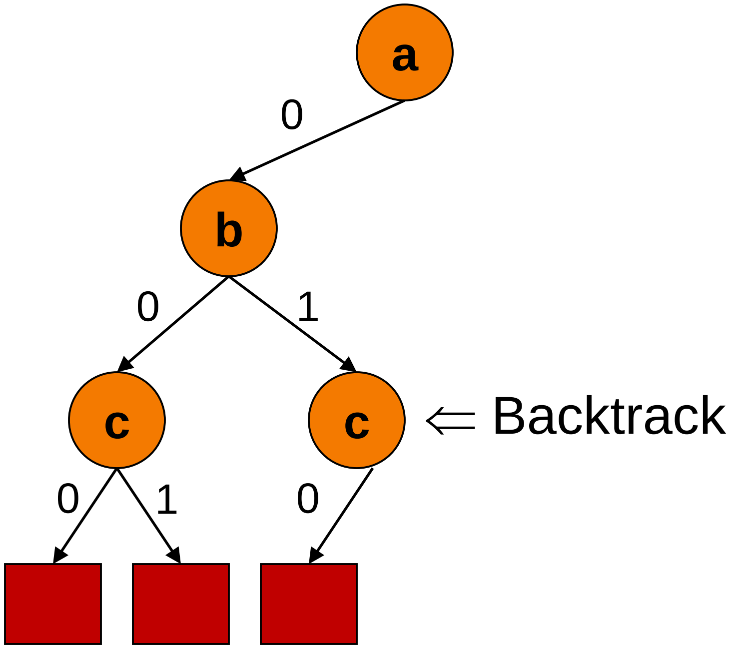


Implication Graph



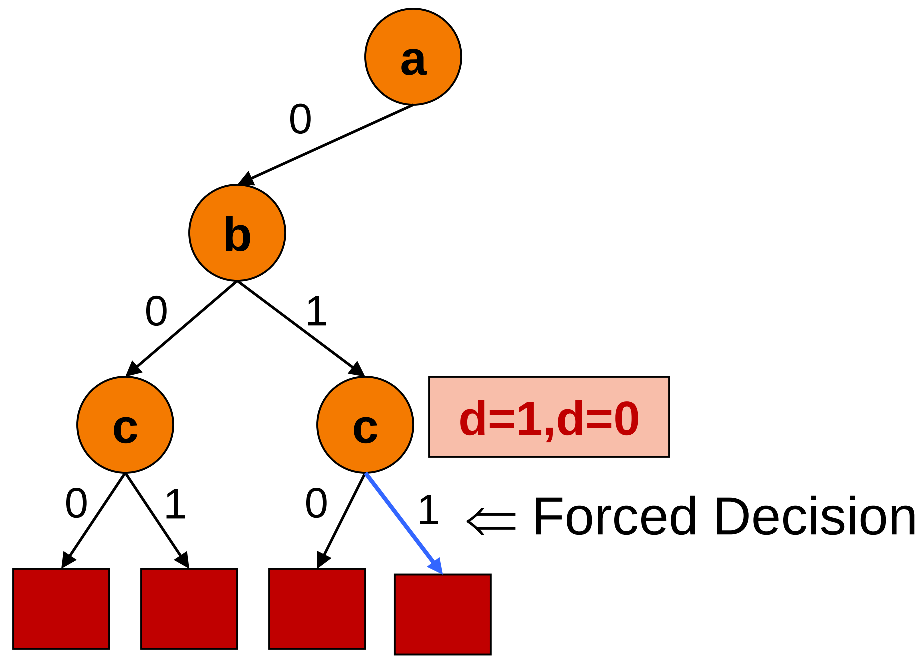
Basic DLL Search

(a' + b + c)
→ (a + c + d)
→ (a + c + d')
→ (a + c' + d)
→ (a + c' + d')
→ (b' + c' + d)
(a' + b + c')
(a' + b' + c)

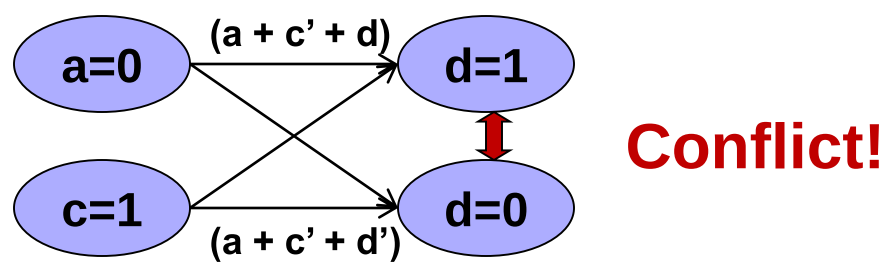


Basic DLL Search

(a’ + b + c)
(a + c + d)
(a + c + d’)
→ (a + c’ + d)
→ (a + c’ + d’)
→ (b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)

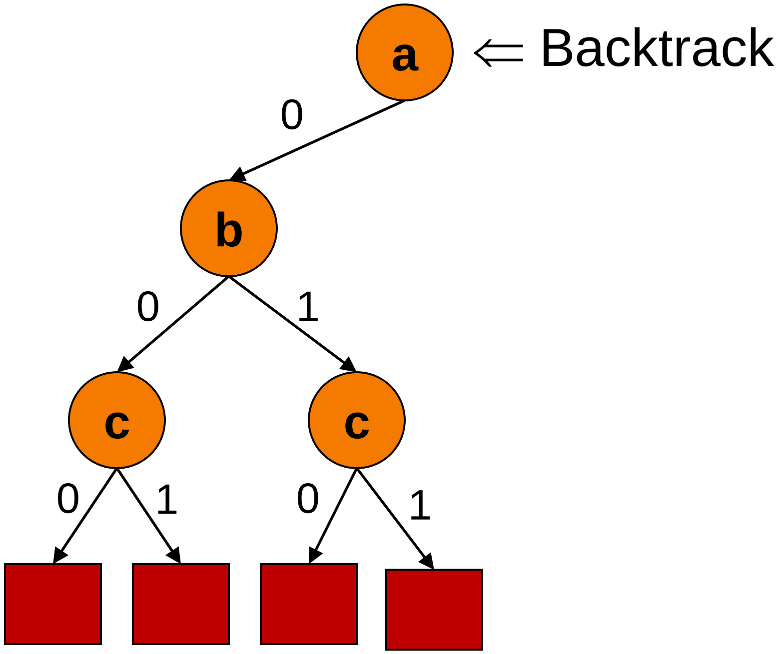


Implication Graph



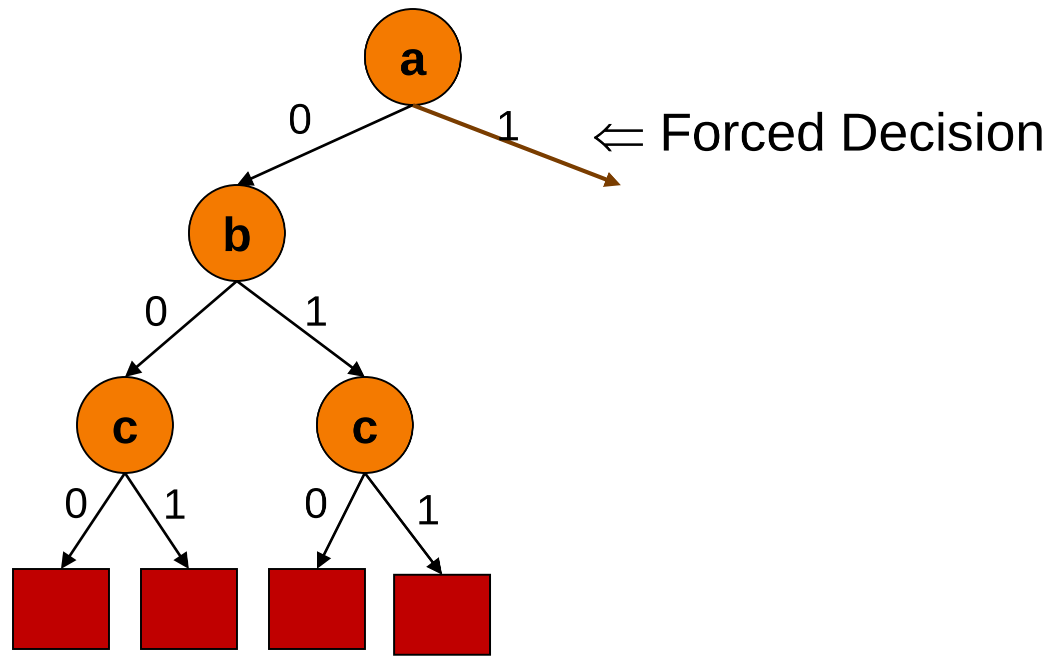
Basic DLL Search

→ (a’ + b + c)
→ (a + c + d)
→ (a + c + d’)
→ (a + c’ + d)
→ (a + c’ + d’)
→ (b’ + c’ + d)
→ (a’ + b + c’)
→ (a’ + b’ + c)



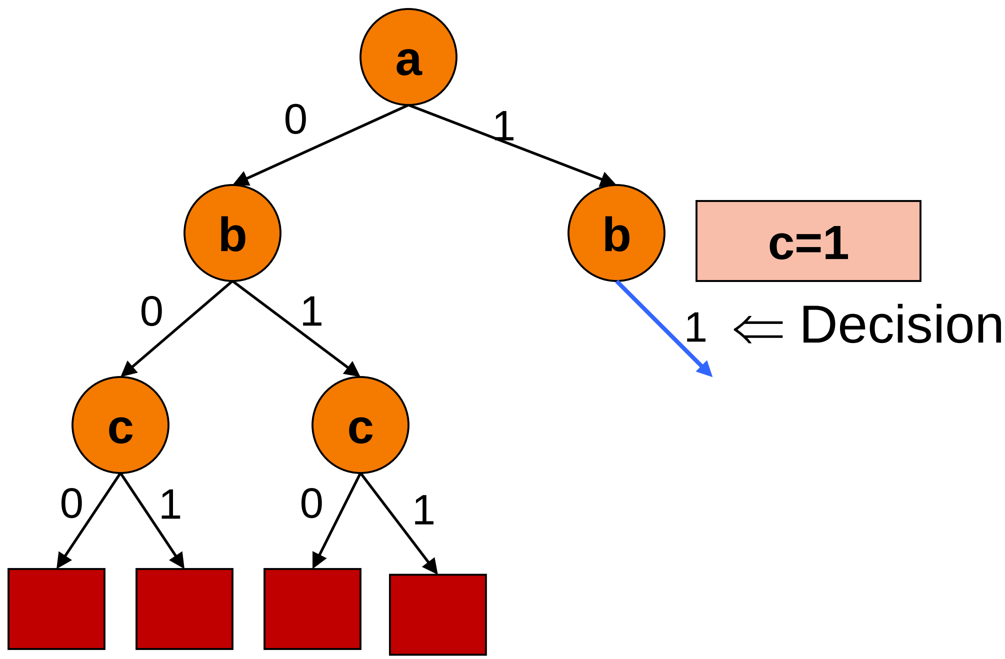
Basic DLL Search

(a' + b + c)
→ (a + c + d)
→ (a + c + d')
→ (a + c' + d)
→ (a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

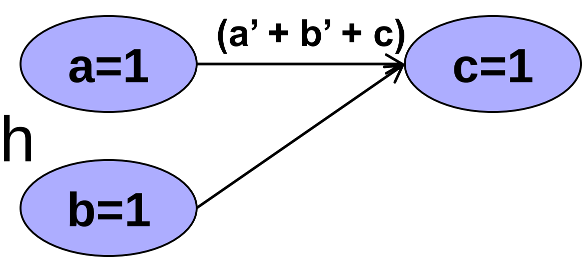


Basic DLL Search

→ (a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
→ (a' + b + c')
→ (a' + b' + c)

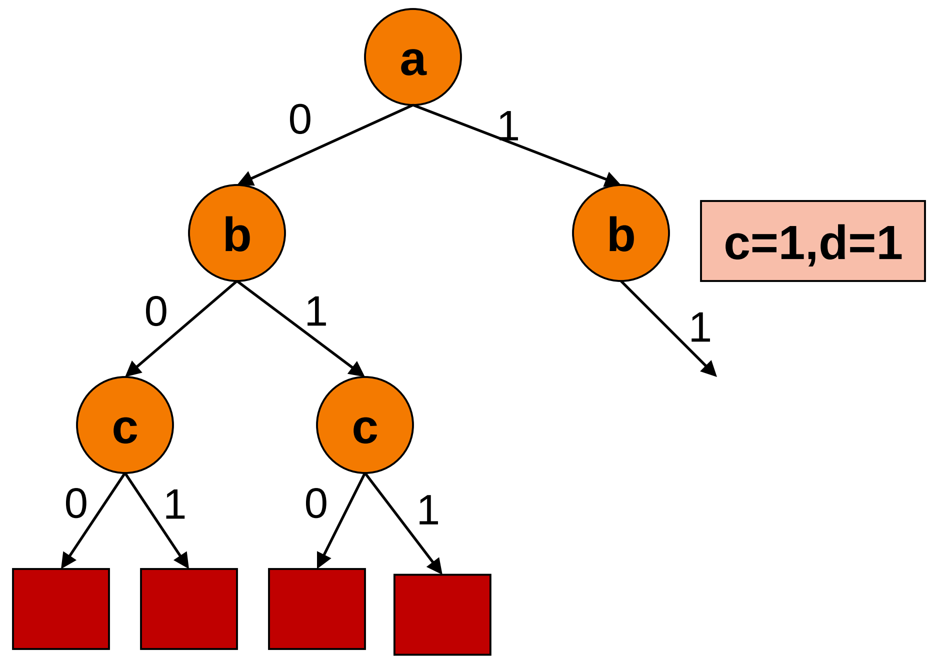


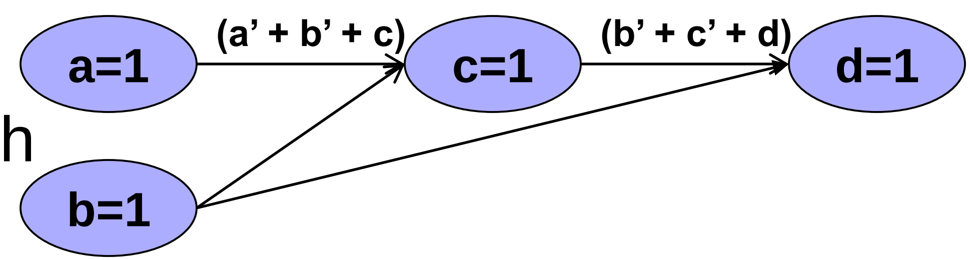
Implication Graph



Basic DLL Search

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
→ (b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)

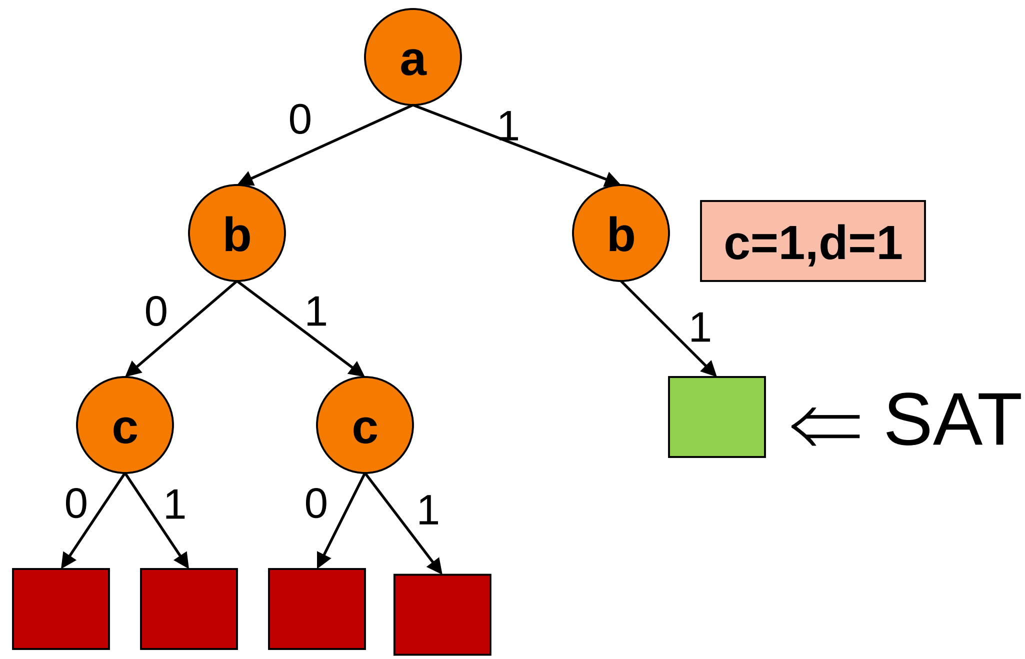




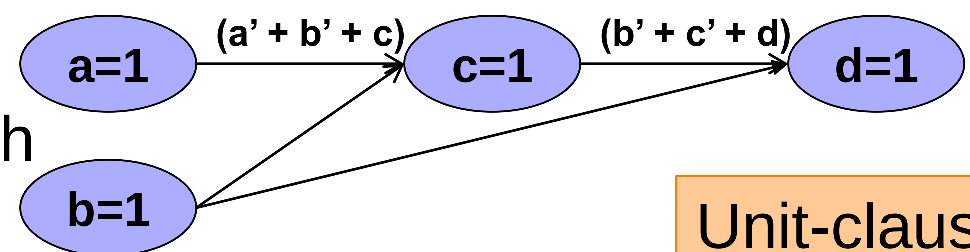
Implication Graph

Basic DLL Search

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
→ (b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



Implication Graph



Unit-clause rule with backtrack search

DPLL SAT Solver

unit clause rule

DPLL(F)
  G ← **BCP**(F)
  if G = ⊤ then return *true*
  if G = ⊥ then return *false*
  p ← **choose**(vars(G))
  return DPLL(G{p ↦ ⊤}) = "SAT" **or** DPLL(G{p ↦ ⊥})

decision heuristics

backtracking search