LOGIC AND THEORETICAL FOUNDATION OF COMPUTER SCIENCE

LaTFoCS

Pamela Fleischmann

fpa@informatik.uni-kiel.de

Winter Semester 2019

Kiel University
Dependable Systems Group

NORMAL FORMS

Motivation

- truth tables are exponentially large

Motivation

- truth tables are exponentially large
- proofs are hard to find

Motivation

- truth tables are exponentially large
- proofs are hard to find
- but a decision procedure for satisfiability is essential!

Motivation

- truth tables are exponentially large
- proofs are hard to find
- but a decision procedure for satisfiability is essential!
- let's restrict ourselves to a subset of formula we can handle better

Conjunctive Normal Form (CNF)

Definition

- C clause: C disjunction of literals

Conjunctive Normal Form (CNF)

Definition

- C clause: C disjunction of literals
- φ formula in CNF: φ is conjunction of clauses

Conjunctive Normal Form (CNF)

Definition

- C clause: C disjunction of literals
- φ formula in CNF: φ is conjunction of clauses

EXAMPLE

$$(p_1 \vee p_3 \vee \neg p_4) \wedge (p_2 \vee \neg p_3) \wedge (\neg p_1 \vee p_2 \vee \neg p_4)$$

Conjunctive Normal Form (CNF)

Definition

- C clause: C disjunction of literals
- φ formula in CNF: φ is conjunction of clauses

EXAMPLE

$$(\overbrace{p_1}^{L_1} \lor \overbrace{p_3}^{L_2} \lor \overbrace{\neg p_4}^{L_3}) \land (\overbrace{p_2}^{L_4} \lor \overbrace{\neg p_3}^{L_2}) \land (\overbrace{\neg p_1}^{L_1} \lor \overbrace{p_2}^{L_4} \lor \overbrace{\neg p_4}^{L_3})$$

Conjunctive Normal Form (CNF)

Definition

- C clause: C disjunction of literals
- φ formula in CNF: φ is conjunction of clauses

EXAMPLE

$$\underbrace{(\overbrace{p_1}^{L_1} \vee \overbrace{p_3}^{L_2} \vee \overbrace{\neg p_4}^{L_3})}_{C_1} \wedge \underbrace{(\overbrace{p_2}^{L_4} \vee \overbrace{\neg p_3}^{L_2})}_{C_2} \wedge \underbrace{(\overbrace{\neg p_1}^{L_1} \vee \overbrace{p_2}^{L_4} \vee \overbrace{\neg p_4}^{L_3})}_{C_3}$$

Advantage of CNF

- φ in CNF if there exist clauses $C_1, \dots, C_n$ with $\varphi = \bigwedge_{i \in [n]} C_i$ for $n \in \mathbb{N}_0$

Advantage of CNF

- φ in CNF if there exist clauses $C_1, \ldots, C_n$ with $\varphi = \bigwedge_{i \in [n]} C_i$ for $n \in \mathbb{N}_0$
- conjunction `true` iff all clauses valuates to `true`, i.e. φ valuates to `true` iff $\forall i \in [n] : C_i$ valuates to `true`

Advantage of CNF

- φ in CNF if there exist clauses $C_1, \dots, C_n$ with $\varphi = \bigwedge_{i\in[n]} C_i$ for $n \in \mathbb{N}_0$
- conjunction `true` iff all clauses valuates to `true`, i.e. φ valuates to `true` iff $\forall i \in [n] :\ C_i$ valuates to `true`
- Notice: $C_1, \dots, C_n$ are not independent but we can start to have a look at them separately

Advantage of CNF

- φ in CNF if there exist clauses $C_1, \ldots, C_n$ with $\varphi = \bigwedge_{i \in [n]} C_i$ for $n \in \mathbb{N}_0$
- conjunction `true` iff all clauses valuates to `true`, i.e. φ valuates to `true` iff $\forall i \in [n] :\ C_i$ valuates to `true`
- Notice: $C_1, \ldots, C_n$ are not independent but we can start to have a look at them separately
- clauses are disjunction which valuate to `true` if at least one literal is `true`

Example 1

$$\underbrace{(\overbrace{p_1}^{L_1} \vee \overbrace{p_3}^{L_2} \vee \overbrace{\neg p_4}^{L_3})}_{C_1} \wedge \underbrace{(\overbrace{p_2}^{L_4} \vee \overbrace{\neg p_3}^{L_2})}_{C_2} \wedge \underbrace{(\overbrace{\neg p_1}^{L_1} \vee \overbrace{p_2}^{L_4} \vee \overbrace{\neg p_4}^{L_3})}_{C_3}$$

Valuation:

1. assume a valuation β with $\beta(p_1) = \texttt{true}$

Example 1

$$\underbrace{(\overbrace{p_1}^{L_1} \vee \overbrace{p_3}^{L_2} \vee \overbrace{\neg p_4}^{L_3})}_{C_1} \wedge \underbrace{(\overbrace{p_2}^{L_4} \vee \overbrace{\neg p_3}^{L_2})}_{C_2} \wedge \underbrace{(\overbrace{\neg p_1}^{L_1} \vee \overbrace{p_2}^{L_4} \vee \overbrace{\neg p_4}^{L_3})}_{C_3}$$

Valuation:

1. assume a valuation β with $\beta(p_1) = \texttt{true}$
2. $\rightsquigarrow$ C_1 valuates to `true`, $\neg p_1$ in C_3 valuates to `false`

Example 1

$$\underbrace{(\overbrace{p_1}^{L_1} \vee \overbrace{p_3}^{L_2} \vee \overbrace{\neg p_4}^{L_3})}_{C_1} \wedge \underbrace{(\overbrace{p_2}^{L_4} \vee \overbrace{\neg p_3}^{L_2})}_{C_2} \wedge \underbrace{(\overbrace{\neg p_1}^{L_1} \vee \overbrace{p_2}^{L_4} \vee \overbrace{\neg p_4}^{L_3})}_{C_3}$$

Valuation:

1. assume a valuation β with $\beta(p_1) = \texttt{true}$
2. $\rightsquigarrow$ C_1 valuates to `true`, $\neg p_1$ in C_3 valuates to `false`
3. assume $\beta(p_2) = \texttt{false}$

Example 1

$$\underbrace{(\overbrace{p_1}^{L_1} \lor \overbrace{p_3}^{L_2} \lor \overbrace{\neg p_4}^{L_3})}_{C_1} \land \underbrace{(\overbrace{p_2}^{L_4} \lor \overbrace{\neg p_3}^{L_2})}_{C_2} \land \underbrace{(\overbrace{\neg p_1}^{L_1} \lor \overbrace{p_2}^{L_4} \lor \overbrace{\neg p_4}^{L_3})}_{C_3}$$

Valuation:

1. assume a valuation β with $\beta(p_1) = \texttt{true}$
2. $\rightsquigarrow$ C_1 valuates to `true`, $\neg p_1$ in C_3 valuates to `false`
3. assume $\beta(p_2) = \texttt{false}$
4. $\rightsquigarrow$ C_2 only valuates to `true` if $\beta(\neg p_3) = \texttt{true}$, thus $\beta(p_3) = \texttt{false}$

Example 1

$$\underbrace{(\overbrace{p_1}^{L_1} \vee \overbrace{p_3}^{L_2} \vee \overbrace{\neg p_4}^{L_3})}_{C_1} \wedge \underbrace{(\overbrace{p_2}^{L_4} \vee \overbrace{\neg p_3}^{L_2})}_{C_2} \wedge \underbrace{(\overbrace{\neg p_1}^{L_1} \vee \overbrace{p_2}^{L_4} \vee \overbrace{\neg p_4}^{L_3})}_{C_3}$$

Valuation:

1. assume a valuation β with $\beta(p_1) = \texttt{true}$
2. $\rightsquigarrow$ C_1 valuates to `true`, $\neg p_1$ in C_3 valuates to `false`
3. assume $\beta(p_2) = \texttt{false}$
4. $\rightsquigarrow$ C_2 only valuates to `true` if $\beta(\neg p_3) = \texttt{true}$, thus $\beta(p_3) = \texttt{false}$
5. for C_3 valuating to `true`, $\neg p_4$ has to valuate to `true`, thus $\beta(p_4) = \texttt{false}$

Example 2

$$\underbrace{(p_1 \vee p_2)}_{C_1} \wedge \underbrace{(\neg p_1 \vee p_2)}_{C_2} \wedge \underbrace{(p_1 \vee \neg p_2)}_{C_3} \wedge \underbrace{(\neg p_1 \vee \neg p_2)}_{C_4}$$

Example 2

$$\underbrace{(p_1 \vee p_2)}_{C_1} \wedge \underbrace{(\neg p_1 \vee p_2)}_{C_2} \wedge \underbrace{(p_1 \vee \neg p_2)}_{C_3} \wedge \underbrace{(\neg p_1 \vee \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{true}$

Example 2

$$\underbrace{(p_1 \vee p_2)}_{C_1} \wedge \underbrace{(\neg p_1 \vee p_2)}_{C_2} \wedge \underbrace{(p_1 \vee \neg p_2)}_{C_3} \wedge \underbrace{(\neg p_1 \vee \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{true}$
2. $\rightsquigarrow$ valuation of C_1, C_3 `true`

Example 2

$$\underbrace{(p_1 \lor p_2)}_{C_1} \land \underbrace{(\neg p_1 \lor p_2)}_{C_2} \land \underbrace{(p_1 \lor \neg p_2)}_{C_3} \land \underbrace{(\neg p_1 \lor \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{true}$
2. $\rightsquigarrow$ valuation of C_1, C_3 `true`
3. for C_2 valuating to `true`, p_2 needs to valuate to `true`

Example 2

$$\underbrace{(p_1 \vee p_2)}_{C_1} \wedge \underbrace{(\neg p_1 \vee p_2)}_{C_2} \wedge \underbrace{(p_1 \vee \neg p_2)}_{C_3} \wedge \underbrace{(\neg p_1 \vee \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{true}$
2. $\rightsquigarrow$ valuation of C_1, C_3 `true`
3. for C_2 valuating to `true`, p_2 needs to valuate to `true`
4. but for C_4 valuating to `true`, p_2 needs to valuate to `false`

Example 2

$$\underbrace{(p_1 \vee p_2)}_{C_1} \wedge \underbrace{(\neg p_1 \vee p_2)}_{C_2} \wedge \underbrace{(p_1 \vee \neg p_2)}_{C_3} \wedge \underbrace{(\neg p_1 \vee \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{true}$
2. $\rightsquigarrow$ valuation of C_1, C_3 `true`
3. for C_2 valuating to `true`, p_2 needs to valuate to `true`
4. but for C_4 valuating to `true`, p_2 needs to valuate to `false`
5. $\rightsquigarrow$ we have a problem if p_1 valuates to `true` (only assumption we made)

Example 2

$$\underbrace{(p_1 \vee p_2)}_{C_1} \wedge \underbrace{(\neg p_1 \vee p_2)}_{C_2} \wedge \underbrace{(p_1 \vee \neg p_2)}_{C_3} \wedge \underbrace{(\neg p_1 \vee \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{false}$

Example 2

$$\underbrace{(p_1 \vee p_2)}_{C_1} \wedge \underbrace{(\neg p_1 \vee p_2)}_{C_2} \wedge \underbrace{(p_1 \vee \neg p_2)}_{C_3} \wedge \underbrace{(\neg p_1 \vee \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{false}$
2. $\rightsquigarrow$ valuation of C_2, C_4 `true`

Example 2

$$\underbrace{(p_1 \lor p_2)}_{C_1} \land \underbrace{(\neg p_1 \lor p_2)}_{C_2} \land \underbrace{(p_1 \lor \neg p_2)}_{C_3} \land \underbrace{(\neg p_1 \lor \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{false}$
2. $\rightsquigarrow$ valuation of C_2, C_4 `true`
3. for C_1 valuating to `true`, p_2 needs to valuate to `true`

Example 2

$$\underbrace{(p_1 \lor p_2)}_{C_1} \land \underbrace{(\neg p_1 \lor p_2)}_{C_2} \land \underbrace{(p_1 \lor \neg p_2)}_{C_3} \land \underbrace{(\neg p_1 \lor \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{false}$
2. $\rightsquigarrow$ valuation of C_2, C_4 `true`
3. for C_1 valuating to `true`, p_2 needs to valuate to `true`
4. but for C_3 valuating to `true`, p_2 needs to valuate to `false`

Example 2

$$\underbrace{(p_1 \vee p_2)}_{C_1} \wedge \underbrace{(\neg p_1 \vee p_2)}_{C_2} \wedge \underbrace{(p_1 \vee \neg p_2)}_{C_3} \wedge \underbrace{(\neg p_1 \vee \neg p_2)}_{C_4}$$

1. assume a valuation β with $\beta(p_1) = \texttt{false}$
2. $\rightsquigarrow$ valuation of C_2, C_4 `true`
3. for C_1 valuating to `true`, p_2 needs to valuate to `true`
4. but for C_3 valuating to `true`, p_2 needs to valuate to `false`
5. $\rightsquigarrow$ also problem if p_1 valuates to `false` (formula is contradiction)

Validity of Disjunctions

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Validity of Disjunctions

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Proof of ←:

- assume there exist $i, j \in [m]$ with $L_i = \neg L_j$, choose them

Validity of Disjunctions

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ valid iff there exist $i, j \in [m]$ with $L_i = \neg L_j$

Proof of ←:

- assume there exist $i, j \in [m]$ with $L_i = \neg L_j$, choose them
- thus $L_1 \vee \cdots \vee L_m =$ $L_1 \vee L_{i-1} \vee \neg L_j \vee L_{i+1} \vee \cdots \vee L_{j-1} \vee L_j \vee L_{j+1} \vee \cdots \vee L_m$

Validity of Disjunctions

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Proof of ←:

- assume there exist $i, j \in [m]$ with $L_i = \neg L_j$, choose them
- thus $L_1 \vee \cdots \vee L_m =$
  $L_1 \vee L_{i-1} \vee \neg L_j \vee L_{i+1} \vee \cdots \vee L_{j-1} \vee L_j \vee L_{j+1} \vee \cdots \vee L_m$
- $L_1 \vee \cdots \vee L_m =$
  $L_j \vee \neg L_j \vee L_1 \vee L_{i-1} \vee L_{i+1} \vee \cdots \vee L_{j-1} \vee L_{j+1} \vee \cdots \vee L_m$

Validity of Disjunctions

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Proof of ←:

- assume there exist $i, j \in [m]$ with $L_i = \neg L_j$, choose them
- thus $L_1 \vee \cdots \vee L_m =$
  $L_1 \vee L_{i-1} \vee \neg L_j \vee L_{i+1} \vee \cdots \vee L_{j-1} \vee L_j \vee L_{j+1} \vee \cdots \vee L_m$
- $L_1 \vee \cdots \vee L_m =$
  $L_j \vee \neg L_j \vee L_1 \vee L_{i-1} \vee L_{i+1} \vee \cdots \vee L_{j-1} \vee L_{j+1} \vee \cdots \vee L_m$
- $L_j \vee \neg L_j$ is valid, i.e. `true` under all valuation

Validity of Disjunctions

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Proof of ←:

- assume there exist $i, j \in [m]$ with $L_i = \neg L_j$, choose them
- thus $L_1 \vee \cdots \vee L_m =$ $L_1 \vee L_{i-1} \vee \neg L_j \vee L_{i+1} \vee \cdots \vee L_{j-1} \vee L_j \vee L_{j+1} \vee \cdots \vee L_m$
- $L_1 \vee \cdots \vee L_m =$ $L_j \vee \neg L_j \vee L_1 \vee L_{i-1} \vee L_{i+1} \vee \cdots \vee L_{j-1} \vee L_{j+1} \vee \cdots \vee L_m$
- $L_j \vee \neg L_j$ is valid, i.e. `true` under all valuation
- disjunction valid if at least one literal `true` under all valuations

Validity of Disjunctions

Lemma

Disjunction of literals $L_1 \lor \cdots \lor L_m$ valid iff there exist $i, j \in [m]$ with $L_i = \neg L_j$

Proof of ←:

- assume there exist $i, j \in [m]$ with $L_i = \neg L_j$, choose them
- thus $L_1 \lor \cdots \lor L_m =$
  $L_1 \lor L_{i-1} \lor \neg L_j \lor L_{i+1} \lor \cdots \lor L_{j-1} \lor L_j \lor L_{j+1} \lor \cdots \lor L_m$
- $L_1 \lor \cdots \lor L_m =$
  $L_j \lor \neg L_j \lor L_1 \lor L_{i-1} \lor L_{i+1} \lor \cdots \lor L_{j-1} \lor L_{j+1} \lor \cdots \lor L_m$
- $L_j \lor \neg L_j$ is valid, i.e. `true` under all valuation
- disjunction valid if at least one literal `true` under all valuations
- thus disjuntion valid

Validity of Disjunction Cont

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Validity of Disjunction Cont

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Proof of → by contraposition:

- assume there does not exist $i, j \in [m]$ with $L_i = \neg L_j$

Validity of Disjunction Cont

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ valid iff there exist $i, j \in [m]$ with $L_i = \neg L_j$

Proof of → by contraposition:

- assume there does not exist $i, j \in [m]$ with $L_i = \neg L_j$
- we have to prove: $L_1 \vee \cdots \vee L_m$ not valid, i.e. `false` under at least one valuation

Validity of Disjunction Cont

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Proof of → by contraposition:

- assume there does not exist $i, j \in [m]$ with $L_i = \neg L_j$
- we have to prove: $L_1 \vee \cdots \vee L_m$ not valid, i.e. `false` under at least one valuation
- consider valuation β which assigns

Validity of Disjunction Cont

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Proof of $\rightarrow$ by contraposition:

- assume there does not exist $i, j \in [m]$ with $L_i = \neg L_j$
- we have to prove: $L_1 \vee \cdots \vee L_m$ not valid, i.e. `false` under at least one valuation
- consider valuation β which assigns
    - `true` to all literals L with $L = p_i$ for an atom p_i

Validity of Disjunction Cont

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Proof of → by contraposition:

- assume there does not exist $i, j \in [m]$ with $L_i = \neg L_j$
- we have to prove: $L_1 \vee \cdots \vee L_m$ not valid, i.e. `false` under at least one valuation
- consider valuation β which assigns
  - `true` to all literals L with $L = p_i$ for an atom p_i
  - `false` to all literals L with $L = \neg p_i$ for an atom p_i

Validity of Disjunction Cont

Lemma

Disjunction of literals $L_1 \vee \cdots \vee L_m$ *valid iff there exist* $i, j \in [m]$ *with* $L_i = \neg L_j$

Proof of → by contraposition:

- assume there does not exist $i, j \in [m]$ with $L_i = \neg L_j$
- we have to prove: $L_1 \vee \cdots \vee L_m$ not valid, i.e. `false` under at least one valuation
- consider valuation β which assigns
  - `true` to all literals L with $L = p_i$ for an atom p_i
  - `false` to all literals L with $L = \neg p_i$ for an atom p_i
- thus disjuntion valuates to `false`

Validity of Formulae in CNF

- conjunction valuates to `true` iff all conjuncts evaluates to `true`

Validity of Formulae in CNF

- conjunction valuates to `true` iff all conjuncts evaluates to `true`
- a CNF evaluates to `true` iff all clauses evaluates to `true`

Validity of Formulae in CNF

- conjunction valuates to `true` iff all conjuncts evaluates to `true`
- a CNF evaluates to `true` iff all clauses evaluates to `true`
- a CNF is valid iff it always evaluates to `true`

Validity of Formulae in CNF

- conjunction valuates to `true` iff all conjuncts evaluates to `true`
- a CNF evaluates to `true` iff all clauses evaluates to `true`
- a CNF is valid iff it always evaluates to `true`
- a CNF is valid iff all clauses always evaluates to `true`

Validity of Formulae in CNF

○ conjunction valuates to `true` iff all conjuncts evaluates to `true`

○ a CNF evaluates to `true` iff all clauses evaluates to `true`

○ a CNF is valid iff it always evaluates to `true`

○ a CNF is valid iff all clauses always evaluates to `true`

○ a CNF is valid iff all clauses are valid

Validity of Formulae in CNF

- conjunction valuates to `true` iff all conjuncts evaluates to `true`
- a CNF evaluates to `true` iff all clauses evaluates to `true`
- a CNF is valid iff it always evaluates to `true`
- a CNF is valid iff all clauses always evaluates to `true`
- a CNF is valid iff all clauses are valid
- validity check of a CNF: test all clauses for validity

Validity of Formulae in CNF

- conjunction valuates to `true` iff all conjuncts evaluates to `true`
- a CNF evaluates to `true` iff all clauses evaluates to `true`
- a CNF is valid iff it always evaluates to `true`
- a CNF is valid iff all clauses always evaluates to `true`
- a CNF is valid iff all clauses are valid
- validity check of a CNF: test all clauses for validity
- notice: high costs

Validity of Formulae in CNF

- conjunction valuates to `true` iff all conjuncts evaluates to `true`
- a CNF evaluates to `true` iff all clauses evaluates to `true`
- a CNF is valid iff it always evaluates to `true`
- a CNF is valid iff all clauses always evaluates to `true`
- a CNF is valid iff all clauses are valid
- validity check of a CNF: test all clauses for validity
- notice: high costs
- and how to get a CNF from an arbitrary formula?

Naïve algorithm for Transforming into CNF

Algorithm 1 Naïve algorithm for CNF

Input: propositional logic formula φ with atoms $p_1, \ldots, p_n$ and the truth table

```
for $\ell = 1$ to $2^n$ do
    $\varphi' = \texttt{true}$
    if $\varphi$ evaluates in $\ell^{th}$ line to false then
        $C_\ell = \texttt{false}$
        for $i = 1$ to $n$ do
            if $p_i$ evaluates to true in $\ell^{th}$ line then
                append $\vee\neg p_i$ to $C_\ell$
            else
                append $\vee p_i$ to $C_\ell$
    append $\wedge C_\ell$ to $\varphi'$
```

Example for naïve algorithm

Notice: the algorithm is based on the completeness proof of the propositional logic!

Consider $\varphi = p_1 \to (p_2 \wedge (p_3 \to \neg p_2)$ with the truth table

p_1	p_2	p_3	φ
`false`	`false`	`false`	`true`
`false`	`false`	`true`	`true`
`false`	`true`	`false`	`true`
`false`	`true`	`true`	`true`
`true`	`false`	`false`	`false`
`true`	`false`	`true`	`false`
`true`	`true`	`false`	`true`
`true`	`true`	`true`	`false`

Example for naïve algorithm

Notice: the algorithm is based on the completeness proof of the propositional logic!

Consider $\varphi = p_1 \rightarrow (p_2 \wedge (p_3 \rightarrow \neg p_2))$ with the truth table

p_1	p_2	p_3	φ	
`false`	`false`	`false`	`true`	
`false`	`false`	`true`	`true`	
`false`	`true`	`false`	`true`	
`false`	`true`	`true`	`true`	
`true`	`false`	`false`	`false`	$\neg p_1 \vee p_2 \vee p_3$
`true`	`false`	`true`	`false`	$\neg p_1 \vee p_2 \vee \neg p_3$
`true`	`true`	`false`	`true`	
`true`	`true`	`true`	`false`	$\neg p_1 \vee \neg p_2 \vee \neg p_3$

Example for naïve algorithm

Notice: the algorithm is based on the completeness proof of the propositional logic!

Consider $\varphi = p_1 \rightarrow (p_2 \wedge (p_3 \rightarrow \neg p_2)$ with the truth table

p_1	p_2	p_3	φ	
`false`	`false`	`false`	`true`	
`false`	`false`	`true`	`true`	
`false`	`true`	`false`	`true`	
`false`	`true`	`true`	`true`	
`true`	`false`	`false`	`false`	$\neg p_1 \vee p_2 \vee p_3$
`true`	`false`	`true`	`false`	$\neg p_1 \vee p_2 \vee \neg p_3$
`true`	`true`	`false`	`true`	
`true`	`true`	`true`	`false`	$\neg p_1 \vee \neg p_2 \vee \neg p_3$

thus: $\varphi \equiv (\neg p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee \neg p_3)$
$\wedge(\neg p_1 \vee \neg p_2 \vee \neg p_3)$

Advantages of CNF

○ formula in CNF can be fast and easily checked for validity

Advantages of CNF

- formula in CNF can be fast and easily checked for validity
- every formula in a class of equivalent formulae is valid if one is valid

Advantages of CNF

- formula in CNF can be fast and easily checked for validity
- every formula in a class of equivalent formulae is valid if one is valid
- it is *enough* to find an equivalent formula in CNF and check its validity

Advantages of CNF

- formula in CNF can be fast and easily checked for validity
- every formula in a class of equivalent formulae is valid if one is valid
- it is *enough* to find an equivalent formula in CNF and check its validity
- the previous algorithm needs the truth table which to build is exponential

Advantages of CNF

- formula in CNF can be fast and easily checked for validity
- every formula in a class of equivalent formulae is valid if one is valid
- it is *enough* to find an equivalent formula in CNF and check its validity
- the previous algorithm needs the truth table which to build is exponential
- there does not exist **the** equivalent CNF for a formula (not unique)

Advantages of CNF

- formula in CNF can be fast and easily checked for validity
- every formula in a class of equivalent formulae is valid if one is valid
- it is *enough* to find an equivalent formula in CNF and check its validity
- the previous algorithm needs the truth table which to build is exponential
- there does not exist **the** equivalent CNF for a formula (not unique)
- different measures: number of conjuncts, overall length, etc

Stepwise Approach to Build an Algorithm

- we will build a deterministic algorithm
- we will use the structure of formulae
- w.l.o.g. we will assume that the formulae are implication free (preprocessing)
- w.l.o.g. we will assume that the formulae are in negation normal form (preprocessing)

Preprocessing: implication freedom

With the equivalence $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$ we can calculate an implication free, equivalent formula.

Preprocessing: implication freedom

With the equivalence $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$ we can calculate an implication free, equivalent formula.

Algorithm 2 ImplFree

```
Input: propositional logic formula $\varphi$
if $\varphi$ is a literal then
    return $\varphi$
if $\varphi = \neg\psi$ then
    return $\neg\,\mathrm{ImplFree}(\psi)$
if $\varphi = \psi_1 \vee \psi_2$ then
    return $\mathrm{ImplFree}(\psi_1) \vee \mathrm{ImplFree}(\psi_2)$
if $\varphi = \psi_1 \wedge \psi_2$ then
    return $\mathrm{ImplFree}(\psi_1) \wedge \mathrm{ImplFree}(\psi_2)$
if $\varphi = \psi_1 \rightarrow \psi_2$ then
    return $\neg\,\mathrm{ImplFree}(\psi_1) \vee \mathrm{ImplFree}(\psi_2)$
```

Example

Notice that the algorithm ImplFree has to be applied recursively!

Example

Notice that the algorithm ImplFree has to be applied recursively!

$$
\begin{aligned}
&\mathrm{ImplFree}((p \lor (q \to r)) \to (s \to (p \to t)))\\
&= \neg\, \mathrm{ImplFree}(p \lor (q \to r)) \lor \mathrm{ImplFree}(s \to (p \to t))\\
&= \neg(p \lor \mathrm{ImplFree}(q \to r)) \lor (\neg s \lor \mathrm{ImplFree}(p \to t))\\
&= \neg(p \lor (\neg q \lor r)) \lor (\neg s \lor (\neg p \lor t))\\
&= (\neg p \land (\neg q \lor r)) \lor (\neg s \lor \neg p \lor t)
\end{aligned}
$$

NNF-Algorithm

Algorithm 3 NNF

```
Input: implication-free, propositional logic formula $\varphi$
if $\varphi$ is a literal then
    return $\varphi$
if $\varphi = \neg\neg\psi$ then
    return $\psi$
if $\varphi = \psi_1 \circ \psi_2$ for $\circ \in \{\wedge, \vee\}$ then
    return $\mathrm{NNF}(\psi_1) \circ \mathrm{NNF}(\psi_2)$
if $\varphi = \neg(\psi_1 \wedge \psi_2)$ then
    return $\mathrm{NNF}(\neg\psi_1) \vee \mathrm{NNF}(\neg\psi_2)$
if $\varphi = \psi_1 \vee \psi_2$ then
    return $\mathrm{NNF}(\neg\psi_1) \wedge \mathrm{NNF}(\neg\psi_2)$
```

Negation Normal Form

If we have a CNF for φ can we compute efficiently a CNF for $\neg\varphi$?

Negation Normal Form

If we have a CNF for φ can we compute efficiently a CNF for $\neg\varphi$?

○ nobody knows the answer

On the way to produce a CNF

- by NNF(ImplFree(φ)) we can now assume that our algorithm gets an implication-free formula in negation normal form

On the way to produce a CNF

- by NNF(ImplFree(φ)) we can now assume that our algorithm gets an implication-free formula in negation normal form
- how to transform the rest?

On the way to produce a CNF

- by NNF(ImplFree(φ)) we can now assume that our algorithm gets an implication-free formula in negation normal form
- how to transform the rest?
- reminder: CNF $C_1 \wedge \cdots \wedge C_n$ with $C_i = \ell_{i1} \vee \cdots \vee \ell_{im_i}$

On the way to produce a CNF

- by NNF(ImplFree(φ)) we can now assume that our algorithm gets an implication-free formula in negation normal form
- how to transform the rest?
- reminder: CNF $C_1 \land \cdots \land C_n$ with $C_i = \ell_{i1} \lor \cdots \lor \ell_{im_i}$
- thus if the input formula φ is a conjunction of φ_1 and φ_2 we only need to transform them and concatenate them with an $\land$ in between

On the way to produce a CNF

- by NNF(ImplFree(φ)) we can now assume that our algorithm gets an implication-free formula in negation normal form
- how to transform the rest?
- reminder: CNF $C_1 \wedge \cdots \wedge C_n$ with $C_i = \ell_{i1} \vee \cdots \vee \ell_{im_i}$
- thus if the input formula φ is a conjunction of φ_1 and φ_2 we only need to transform them and concatenate them with an $\wedge$ in between
- how to transform a disjunction into a semantically equivalent conjunction

Transforming a DNF into a CNF

its similar to transform a product of sums into a sum of products:

$$(a + b) \cdot (c + d) = a \cdot c + a \cdot d + b \cdot c + b \cdot d$$

Transforming a DNF into a CNF

its similar to transform a product of sums into a sum of products:

$$(a \wedge b) \vee (c \wedge d) = (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$$

resp.

$$b \vee (c \wedge d) = (b \vee c) \wedge (b \vee d)$$

DNF to CNF

Algorithm 4 DNF2CNF

```
Input: $\varphi = \psi_1 \lor \psi_2$ and $\psi_1, \psi_2$ in CNF
if $\psi_1$ is $\chi_1 \land \chi_2$ then
    return DNF2CNF($\chi_1 \lor \psi_2$) $\land$ DNF2CNF($\chi_2 \lor \psi_2$)
else if $\psi_2 = \chi_1 \land \chi_2$ then
    return DNF2CNF($\psi_1 \lor \chi_1$) $\land$ DNF2CNF($\psi_1 \lor \chi_2$)
else
    return $\psi_1 \lor \psi_2$
```

The CNF-Algorithm

Now we are able to present the algorithm:

Algorithm 5 CNF

```
Input: $\varphi$ implication-free and in NNF
if $\varphi$ literal then
    return $\varphi$
else if $\varphi = \psi_1 \wedge \psi_2$ then
    return CNF($\varphi_1$) $\wedge$ CNF($\varphi_2$)
else if $\varphi = \psi_1 \vee \psi_2$ then
    return DNF2CNF(CNF($\varphi_1$ $\vee$ CNF($\varphi_2$)
```

Conclusion for CNF in general

- transforming an arbitrary formula into a CNF is not efficiently possible

Conclusion for CNF in general

- transforming an arbitrary formula into a CNF is not efficiently possible
- if we have a CNF, validity and satisfiability can be checked more easily than for arbitrary formulae

Conclusion for CNF in general

- transforming an arbitrary formula into a CNF is not efficiently possible
- if we have a CNF, validity and satisfiability can be checked more easily than for arbitrary formulae
- the real word is not build in CNF :/

Conclusion for CNF in general

- transforming an arbitrary formula into a CNF is not efficiently possible
- if we have a CNF, validity and satisfiability can be checked more easily than for arbitrary formulae
- the real word is not build in CNF :/
- how are we thinking?

Conclusion for CNF in general

- transforming an arbitrary formula into a CNF is not efficiently possible
- if we have a CNF, validity and satisfiability can be checked more easily than for arbitrary formulae
- the real word is not build in CNF :/
- how are we thinking?
- often in implications!

Conclusion for CNF in general

- transforming an arbitrary formula into a CNF is not efficiently possible
- if we have a CNF, validity and satisfiability can be checked more easily than for arbitrary formulae
- the real word is not build in CNF :/
- how are we thinking?
- often in implications!
- ... not all implications are easy to handle

Motivation for restricted implications

disjunctions in conclusions are often problematic in normal life:

- if Paula comes to the party Tom and Max are not coming both ($p \rightarrow \neg t \vee \neg m$)
- Tom is allergic to peanuts.
- Max will bring a barbecue.
- Paula cannot decide wether she comes before the very last minute.

Motivation for restricted implications

disjunctions in conclusions are often problematic in normal life:

- if Paula comes to the party Tom and Max are not coming both ($p \rightarrow \neg t \vee \neg m$)
- Tom is allergic to peanuts.
- Max will bring a barbecue.
- Paula cannot decide wether she comes before the very last minute.

well... do we have to organise a barbecue on our own?
can we prepare something with peanuts?

Motivation for restricted implications

disjunctions in conclusions are often problematic in normal life:

- if Paula comes to the party Tom and Max are not coming both ($p \longrightarrow \neg t \vee \neg m$)
- Tom is allergic to peanuts.
- Max will bring a barbecue.
- Paula cannot decide wether she comes before the very last minute.

well... do we have to organise a barbecue on our own?
can we prepare something with peanuts?

- things are easier if we only have one positive atom as a conclusion

Motivation refined

If we now restrict the premise also to a conjunction of positive atoms we get something nice

$$p_1 \wedge \cdots \wedge p_n \longrightarrow q \equiv \neg p_1 \vee \cdots \vee \neg p_n \vee q$$

Motivation refined

If we now restrict the premise also to a conjunction of positive atoms we get something nice

$$p_1 \wedge \cdots \wedge p_n \longrightarrow q \equiv \neg p_1 \vee \cdots \vee \neg p_n \vee q$$

Definition

○ Clause $C = (\ell_1 \vee \ldots \ell_n)$ of literals ℓ_i, $i \in [n]$, $n \in \mathbb{N}$ is a Horn Clause iff there exists exactly one $i \in [n]$ such that ℓ_i is a positive atom.

Motivation refined

If we now restrict the premise also to a conjunction of positive atoms we get something nice

$$p_1 \wedge \cdots \wedge p_n \rightarrow q \equiv \neg p_1 \vee \cdots \vee \neg p_n \vee q$$

Definition

- Clause $C = (\ell_1 \vee \ldots \ell_n)$ of literals ℓ_i, $i \in [n]$, $n \in \mathbb{N}$ is a Horn Clause iff there exists exactly one $i \in [n]$ such that ℓ_i is a positive atom.
- Horn formula: conjunction of Horn clauses (named after american mathematician Alfred Horn (1918-2001))

Motivation refined

If we now restrict the premise also to a conjunction of positive atoms we get something nice

$$p_1 \wedge \cdots \wedge p_n \longrightarrow q \equiv \neg p_1 \vee \cdots \vee \neg p_n \vee q$$

Definition

- Clause $C = (\ell_1 \vee \ldots \ell_n)$ of literals ℓ_i, $i \in [n]$, $n \in \mathbb{N}$ is a Horn Clause iff there exists exactly one $i \in [n]$ such that ℓ_i is a positive atom.
- Horn formula: conjunction of Horn clauses (named after american mathematician Alfred Horn (1918-2001))

- these clauses are very important in logic programming

Satisfiability of Horn Formulae

for Horn formulae there exists a very easy algorithm deciding whether the formula is satisfiable or not:

Satisfiability of Horn Formulae

for Horn formulae there exists a very easy algorithm deciding whether the formula is satisfiable or not:

- for easier descriptions of tautologies and contradictions we introduce $\top$ for `true` and $\bot$ for false as special atoms

Satisfiability of Horn Formulae

for Horn formulae there exists a very easy algorithm deciding whether the formula is satisfiable or not:

- for easier descriptions of tautologies and contradictions we introduce $\top$ for `true` and $\bot$ for false as special atoms
- $\top \rightarrow p$ means p *is always* `true`

Satisfiability of Horn Formulae

for Horn formulae there exists a very easy algorithm deciding whether the formula is satisfiable or not:

- for easier descriptions of tautologies and contradictions we introduce $\top$ for `true` and $\bot$ for false as special atoms
- $\top \rightarrow p$ means p *is always* `true`
- $\psi \rightarrow \bot$ means *if* ψ *holds we have a contradiction*

Satisfiability of Horn Formulae - Algorithm

Algorithm 6 Horn-SAT

```
Input: $\varphi$ Horn formula
mark all occurrences of $\top$
while ($\exists p_1 \wedge \cdots \wedge p_{k_i} \longrightarrow q$ and $\forall j \in [k_i] : p_j$ marked and $q$ unmarked) do
    mark $q$
if ($\bot$ marked) then
    return UNSAT
return SAT
```

Correctness and Completeness of Horn-SAT

Theorem

The algorithm Horn-SAT is correct, complete and the while loop is executed at most $n + 1$ *times for the number of atoms* $n \in \mathbb{N}$ *in the formula.*

Correctness and Completeness of Horn-SAT

Theorem

The algorithm Horn-SAT is correct, complete and the while loop is executed at most $n + 1$ *times for the number of atoms* $n \in \mathbb{N}$ *in the formula.*

Proof:

Correctness and Completeness of Horn-SAT

Theorem

The algorithm Horn-SAT is correct, complete and the while loop is executed at most $n + 1$ times for the number of atoms $n \in \mathbb{N}$ in the formula.

Proof:

- the condition of the while-loop guarantees that the body of the while-loop is only executed if an atom is marked

Correctness and Completeness of Horn-SAT

Theorem

The algorithm Horn-SAT is correct, complete and the while loop is executed at most $n + 1$ times for the number of atoms $n \in \mathbb{N}$ in the formula.

Proof:

- the condition of the while-loop guarantees that the body of the while-loop is only executed if an atom is marked
- there are only n atoms to mark

Correctness and Completeness of Horn-SAT

Theorem

The algorithm Horn-SAT is correct, complete and the while loop is executed at most $n + 1$ times for the number of atoms $n \in \mathbb{N}$ in the formula.

Proof:

- the condition of the while-loop guarantees that the body of the while-loop is only executed if an atom is marked
- there are only n atoms to mark
- thus the while-conditions is checked at most $n + 1$ times

Correctness and Completeness of Horn-SAT

Theorem

The algorithm Horn-SAT is correct, complete and the while loop is executed at most $n + 1$ times for the number of atoms $n \in \mathbb{N}$ in the formula.

Proof:

- the condition of the while-loop guarantees that the body of the while-loop is only executed if an atom is marked
- there are only n atoms to mark
- thus the while-conditions is checked at most $n + 1$ times
- $\rightsquigarrow$ termination (and roughly a run-time)

Proof of Soundness

- idea for marking: marking a variable $\hat{=}$ variable has to be `true` in all valuation satisfying the formula

Proof of Soundness

- idea for marking: marking a variable $\hat{=}$ variable has to be `true` in all valuation satisfying the formula

Claim: At any time the marked variables need to evaluate to `true` in all valuations in which φ evaluated to `true`.

Proof of Soundness

- idea for marking: marking a variable $\hat{=}$ variable has to be `true` in all valuation satisfying the formula

Claim: At any time the marked variables need to evaluate to `true` in all valuations in which φ evaluated to `true`.
Proof by induction on the numbers of while-loop executions:

- Base Case: no execution of the while loop
- thus only $\bot$ is marked with `true`
- $\top$ needs to be `true` by definition

Poof of Soundness Cont

- Induction Hypothesis: after the k^{th} iteration of the while-loop all marked atoms need to be true in all valuations in which φ is `true` for one arbitrary but fixed $k \in \mathbb{N}_0$

Poof of Soundness Cont

- Induction Hypothesis: after the k^{th} iteration of the while-loop all marked atoms need to be true in all valuations in which φ is `true` for one arbitrary but fixed $k \in \mathbb{N}_0$
- Assume the while-loop to be executed $k + 1$ times

Poof of Soundness Cont

- Induction Hypothesis: after the k^{th} iteration of the while-loop all marked atoms need to be true in all valuations in which φ is `true` for one arbitrary but fixed $k \in \mathbb{N}_0$
- Assume the while-loop to be executed $k + 1$ times
- thus there exist a $p_1 \wedge \cdots \wedge p_{k_i} \longrightarrow q$ with all p_j marked and q unmarked which triggers the $(k + 1)^{\text{th}}$ execution

Poof of Soundness Cont

- Induction Hypothesis: after the k^{th} iteration of the while-loop all marked atoms need to be true in all valuations in which φ is `true` for one arbitrary but fixed $k \in \mathbb{N}_0$
- Assume the while-loop to be executed $k+1$ times
- thus there exist a $p_1 \wedge \cdots \wedge p_{k_i} \longrightarrow q$ with all p_j marked and q unmarked which triggers the $(k+1)^{\text{th}}$ execution
- IH $\Rightarrow$ in any valuation in which φ is `true` all these p_j have to be `true`

Poof of Soundness Cont

- Induction Hypothesis: after the k^{th} iteration of the while-loop all marked atoms need to be true in all valuations in which φ is `true` for one arbitrary but fixed $k \in \mathbb{N}_0$
- Assume the while-loop to be executed $k + 1$ times
- thus there exist a $p_1 \land \cdots \land p_{k_i} \to q$ with all p_j marked and q unmarked which triggers the $(k + 1)^{\text{th}}$ execution
- IH $\Rightarrow$ in any valuation in which φ is `true` all these p_j have to be `true`
- by truth table q has to be true in these valuation since otherwise the implication (conjunct of φ) would be `false` and thus also φ

Proof of Completeness

- $\perp$ marked means there exists an implication which leads to `false`

Proof of Completeness

- $\perp$ marked means there exists an implication which leads to `false`
- the while-condition ensure that the premise is `true`

Proof of Completeness

- $\perp$ marked means there exists an implication which leads to `false`
- the while-condition ensure that the premise is `true`
- thus the implication is `false` and so the conjunct

Proof of Completeness

- $\perp$ marked means there exists an implication which leads to `false`
- the while-condition ensure that the premise is `true`
- thus the implication is `false` and so the conjunct
- thus φ is not satisfiable

Proof of Completeness Cont

- if $\bot$ is not marked assign to all marked atoms `true` and to all other `false`

Proof of Completeness Cont

- if $\bot$ is not marked assign to all marked atoms `true` and to all other `false`
- Suppose: φ is not `true`

Proof of Completeness Cont

- if $\perp$ is not marked assign to all marked atoms `true` and to all other `false`
- Suppose: φ is not `true`
    - one of the conjuncts is `false`

Proof of Completeness Cont

- if $\bot$ is not marked assign to all marked atoms `true` and to all other `false`
- Suppose: φ is not `true`
  - one of the conjuncts is `false`
  - Horn clause means conjunct is implication

Proof of Completeness Cont

- if $\perp$ is not marked assign to all marked atoms `true` and to all other `false`
- Suppose: φ is not `true`
  - one of the conjuncts is `false`
  - Horn clause means conjunct is implication
  - implication only `false` if premise `true` and conclusion `false`

Proof of Completeness Cont

- if $\bot$ is not marked assign to all marked atoms `true` and to all other `false`
- Suppose: φ is not `true`
  - one of the conjuncts is `false`
  - Horn clause means conjunct is implication
  - implication only `false` if premise `true` and conclusion `false`
  - while-statement: premise `true` implies to mark the conclusion

Proof of Completeness Cont

- if $\perp$ is not marked assign to all marked atoms `true` and to all other `false`
- Suppose: φ is not `true`
  - one of the conjuncts is `false`
  - Horn clause means conjunct is implication
  - implication only `false` if premise `true` and conclusion `false`
  - while-statement: premise `true` implies to mark the conclusion
  - thus: conclusion cannot be `false`

Proof of Completeness Cont

- if $\bot$ is not marked assign to all marked atoms `true` and to all other `false`
- Suppose: φ is not `true`
  - one of the conjuncts is `false`
  - Horn clause means conjunct is implication
  - implication only `false` if premise `true` and conclusion `false`
  - while-statement: premise `true` implies to mark the conclusion
  - thus: conclusion cannot be `false`
- φ `true` $\square$

Motivation for more general SAT-Solver

- Idea of Horn-SAT algorithm: markings are a constraint

Motivation for more general SAT-Solver

- Idea of Horn-SAT algorithm: markings are a constraint
- we save with the marking knowledge about which atoms have to be set to `true` for the formula being evaluated to `true`

$$p \land (q \lor r) \rightsquigarrow p \text{ needs to be } \texttt{true}$$

Motivation for more general SAT-Solver

- Idea of Horn-SAT algorithm: markings are a constraint
- we save with the marking knowledge about which atoms have to be set to `true` for the formula being evaluated to `true`

$$p \wedge (q \vee r) \rightsquigarrow p \text{ needs to be } \texttt{true}$$

- on the other hand

$$\neg p \wedge (q \vee r) \rightsquigarrow p \text{ needs to be } \texttt{false}$$

Motivation for more general SAT-Solver

- Idea of Horn-SAT algorithm: markings are a constraint
- we save with the marking knowledge about which atoms have to be set to `true` for the formula being evaluated to `true`

$$p \land (q \lor r) \rightsquigarrow p \text{ needs to be } \texttt{true}$$

- on the other hand

$$\neg p \land (q \lor r) \rightsquigarrow p \text{ needs to be } \texttt{false}$$

- and even more generalised: in a conjunction each conjunct needs to be true

Motivation for more general SAT-Solver

- Idea of Horn-SAT algorithm: markings are a constraint
- we save with the marking knowledge about which atoms have to be set to `true` for the formula being evaluated to `true`

$$p \wedge (q \vee r) \rightsquigarrow p \text{ needs to be } \texttt{true}$$

- on the other hand

$$\neg p \wedge (q \vee r) \rightsquigarrow p \text{ needs to be } \texttt{false}$$

- and even more generalised: in a conjunction each conjunct needs to be true
- we capture these marking generalisation in the following algorithm

A linear solver: Translating formula into adequate fragment

- $T(p) = p$
- $T(\neg\varphi) = \neg T(\varphi)$
- $T(\varphi_1 \wedge \varphi_2) = T(\varphi_1) \wedge T(\varphi_2)$
- $T(\varphi_1 \vee \varphi_2) = \neg(\neg T(\varphi_1) \wedge \neg T(\varphi_2))$
- $T(\varphi_1 \rightarrow \varphi_2) = \neg(T(\varphi_1) \wedge \neg T(\varphi_2))$

A linear solver: Translating formula into adequate fragment

- $T(p) = p$
- $T(\neg\varphi) = \neg T(\varphi)$
- $T(\varphi_1 \wedge \varphi_2) = T(\varphi_1) \wedge T(\varphi_2)$
- $T(\varphi_1 \vee \varphi_2) = \neg(\neg T(\varphi_1) \wedge \neg T(\varphi_2))$
- $T(\varphi_1 \rightarrow \varphi_2) = \neg(T(\varphi_1) \wedge \neg T(\varphi_2))$

after applying T a formula only contains atoms, $\neg$ and $\wedge$

A linear solver: Example

$$\varphi = p \wedge (q \longrightarrow \neg p)$$

A linear solver: Example

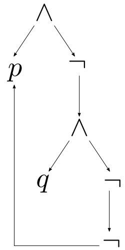
$\varphi = p \wedge (q \longrightarrow \neg p)$

- we have to transform it

$$\begin{aligned} T(\varphi) &= T(p) \wedge T(q \longrightarrow \neg p) \\ &= p \wedge \neg(T(q) \wedge \neg T(\neg p)) \\ &= p \wedge \neg(q \wedge \neg(\neg T(p)) \\ &= p \wedge \neg(q \wedge \neg(\neg p)) \end{aligned}$$

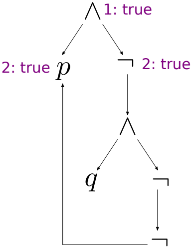
A linear solver: DAG

directed acyclic graph (DAG) for $p \land \neg(q \land \neg(\neg p))$



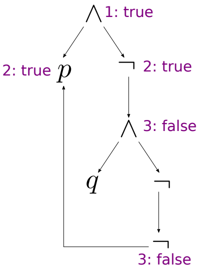
A linear solver: DAG

we now analyse the graph:



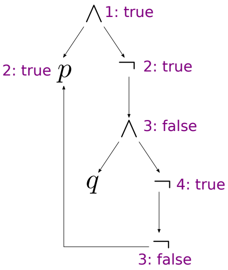
A linear solver: DAG

we now analyse the graph:



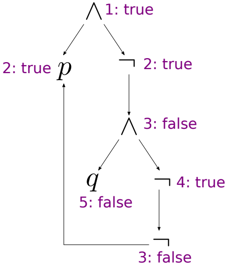
A linear solver: DAG

we now analyse the graph:



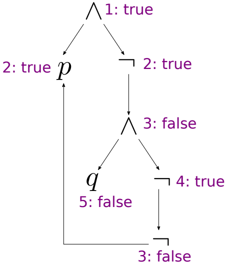
A linear solver: DAG

we now analyse the graph:



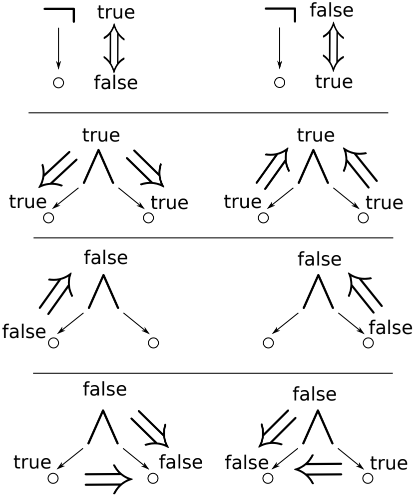
A linear solver: DAG

we now analyse the graph:



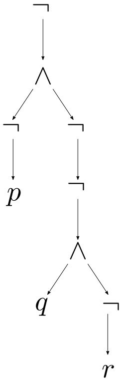
Thus the formula is satisfiable

A linear solver: Forcing Laws



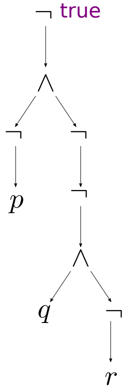
A linear solver: Counter-Example

directed acyclic graph (DAG) for $\neg(\neg p \wedge \neg(\neg(q \wedge \neg r)))$



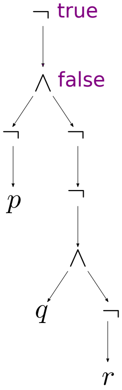
A linear solver: Counter-Example

we now analyse the graph:



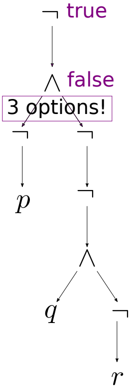
A linear solver: Counter-Example

we now analyse the graph:



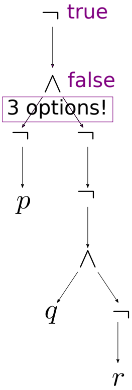
A linear solver: Counter-Example

we now analyse the graph:



A linear solver: Counter-Example

we now analyse the graph:



The linearity comes to the costs of not being able to solve this problem!

New Approach - Tracking down the problem

○ Examining the linear approach:

New Approach - Tracking down the problem

- Examining the linear approach:
    1. all nodes in the DAG marked (each edge once taken) ⇒ marking of the atoms is a satisfying marking of the formula

New Approach - Tracking down the problem

- Examining the linear approach:
  1. all nodes in the DAG marked (each edge once taken) ⇒ marking of the atoms is a satisfying marking of the formula
  2. all nodes in the DAG marked but a `true` marking was overwritten with a `false` or v.v. ⇒ the formula is unsatisfiable

New Approach - Tracking down the problem

- Examining the linear approach:
  1. all nodes in the DAG marked (each edge once taken) ⇒ marking of the atoms is a satisfying marking of the formula
  2. all nodes in the DAG marked but a `true` marking was overwritten with a `false` or v.v. ⇒ the formula is unsatisfiable
- Problem: not all nodes marked (as seen in the example)

New Approach - Guessing

- we run the linear solver and get a not completely marked DAG

New Approach - Guessing

- we run the linear solver and get a not completely marked DAG
- for each remaining node:

New Approach - Guessing

- we run the linear solver and get a not completely marked DAG
- for each remaining node:
    - pick one

New Approach - Guessing

- we run the linear solver and get a not completely marked DAG
- for each remaining node:
  - pick one
  - mark this node temporarily with `true` and propagate

New Approach - Guessing

- we run the linear solver and get a not completely marked DAG
- for each remaining node:
    - pick one
    - mark this node temporarily with `true` and propagate
        - if you find contradictions, you know that the node cannot be marked with `true`

New Approach - Guessing

- we run the linear solver and get a not completely marked DAG
- for each remaining node:
  - pick one
  - mark this node temporarily with `true` and propagate
    - if you find contradictions, you know that the node cannot be marked with `true`
  - mark this node **also** with `false` and propagate

New Approach - Guessing

- we run the linear solver and get a not completely marked DAG
- for each remaining node:
    - pick one
    - mark this node temporarily with `true` and propagate
        - if you find contradictions, you know that the node cannot be marked with `true`
    - mark this node **also** with `false` and propagate
        - if you find contradictions, you can output UNSAT

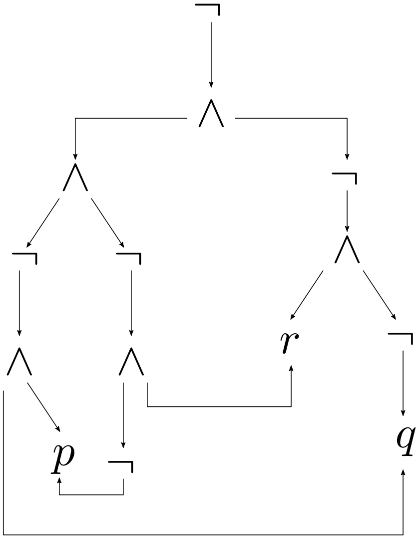
New Approach - Guessing

- we run the linear solver and get a not completely marked DAG
- for each remaining node:
  - pick one
  - mark this node temporarily with `true` and propagate
    - if you find contradictions, you know that the node cannot be marked with `true`
  - mark this node **also** with `false` and propagate
    - if you find contradictions, you can output UNSAT
    - if not search for nodes with the same temporary marking in the `true` and the `false` case and turn them to permanent markings

New Approach - Guessing

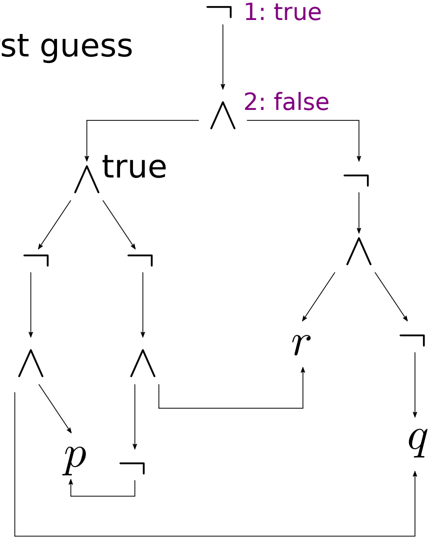
- we run the linear solver and get a not completely marked DAG
- for each remaining node:
  - pick one
  - mark this node temporarily with `true` and propagate
    - if you find contradictions, you know that the node cannot be marked with `true`
  - mark this node **also** with `false` and propagate
    - if you find contradictions, you can output UNSAT
    - if not search for nodes with the same temporary marking in the `true` and the `false` case and turn them to permanent markings
  - delete the temporary markings

Example

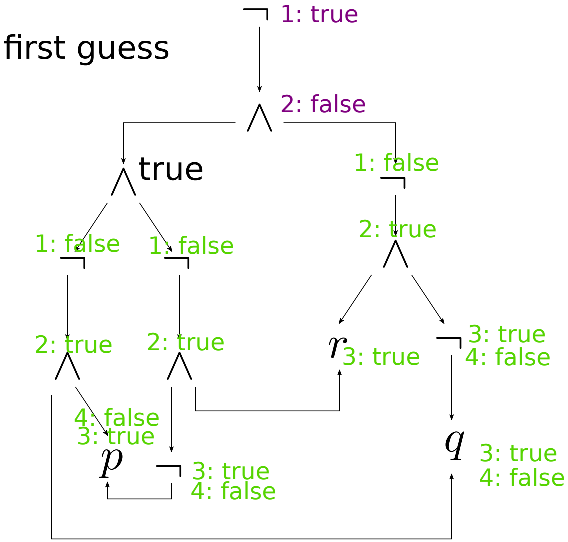


Example

first guess

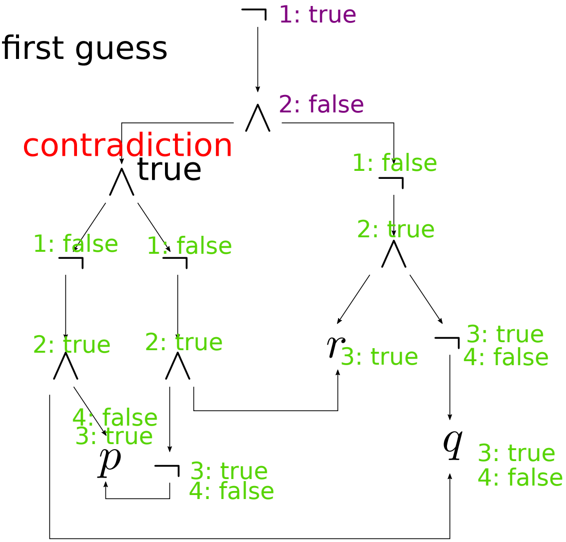


Example

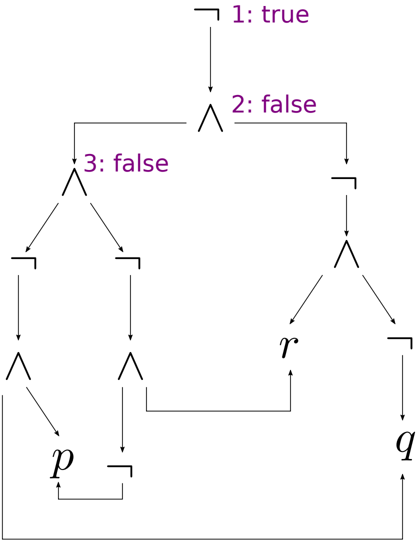

first guess

Example

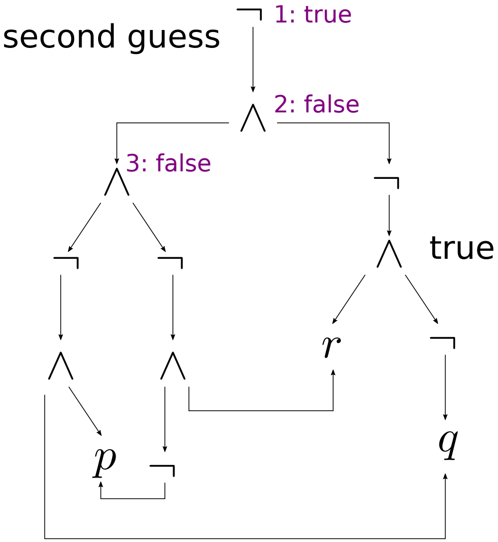
first guess



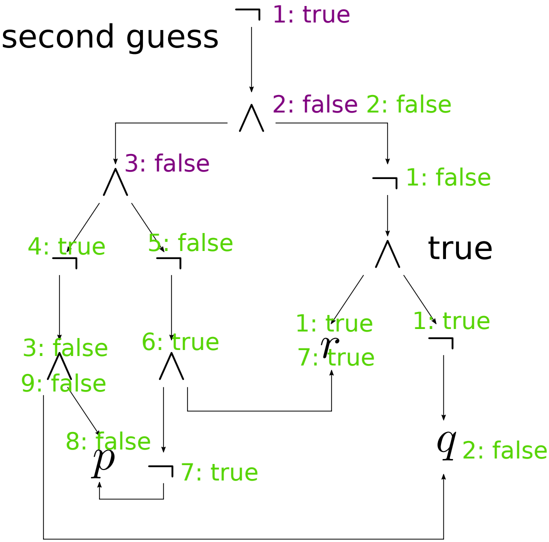
Example



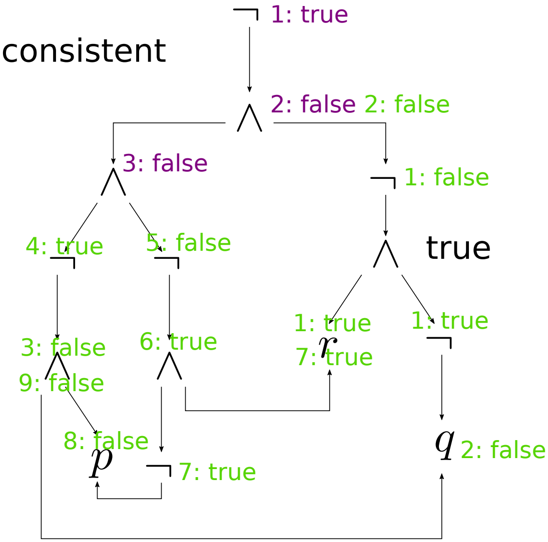
Example



Example


second guess
¬ 1: true
∧ 2: false 2: false
∧ 3: false
¬ 1: false
4: true ¬
5: false ¬
∧ true
3: false ∧ 9: false
6: true ∧
1: true r 7: true
1: true ¬
8: false p
¬ 7: true
q 2: false

Example


¬ 1: true
consistent
∧ 2: false 2: false
∧ 3: false
¬ 1: false
4: true ¬
5: false ¬
∧ true
3: false ∧ 9: false
6: true ∧
1: true r 7: true
1: true ¬
8: false p
¬ 7: true
q 2: false