

Logic: Propositional Logic (Part II)

Alessandro Artale

Free University of Bozen-Bolzano
Faculty of Computer Science
<http://www.inf.unibz.it/~artale>

Discrete Mathematics and Logic — BSc course

Thanks to Prof. Enrico Franconi for providing the slides

Decision Procedures in Logic: Soundness

A **decision procedure** solves a problem with YES or NO answers:

$$KB \vdash_i \alpha$$

- Sentence α can be derived from the set of sentences KB by procedure i .
- *Soundness*: procedure i is sound if whenever procedure i proves that a sentence α can be derived from a set of sentences KB ($KB \vdash_i \alpha$), then it is also true that KB entails α ($KB \models \alpha$).
 - “no wrong inferences are drawn”
 - A sound procedure may fail to find the solution in some cases, when there is actually one.

Decision Procedures in Logic: Completeness

A **decision procedure** solves a problem with YES or NO answers:

$$KB \vdash_i \alpha$$

- Sentence α can be derived from the set of sentences KB by procedure i .
- *Completeness*: procedure i is complete if whenever a set of sentences KB entails a sentence α ($KB \models \alpha$), then procedure i proves that α can be derived from KB ($KB \vdash_i \alpha$).
 - “all the correct inferences are drawn”
 - A complete procedure may claim to have found a solution in some cases, when there is actually no solution.

Sound and Incomplete Algorithms

- Sound and incomplete algorithms are very popular: they are considered *good approximations* of problem solving procedures.
- Sound and incomplete algorithms may reduce the algorithm complexity.
- Sound and incomplete algorithms are often used due to the inability of programmers to find sound and complete algorithms.

Good Decision procedures

- If an incomplete reasoning mechanism is provided, we can conclude either that the semantics of the representation language does not really capture the meaning of the “world” and of “*what should follow*”, or that the algorithms can not infer all the things we would expect.
- Having **sound and complete** reasoning procedures is important!
- Sound and complete decision procedures are good candidates for implementing reasoning modules within larger applications.

An extreme example

Let's consider two decision procedures:

- F , which always returns the result **NO** independently from its input
- T , which always returns the result **YES** independently from its input

An extreme example

Let's consider two decision procedures:

- F , which always returns the result **NO** independently from its input
- T , which always returns the result **YES** independently from its input

Let's consider the problem of computing entailment between formulas:

- F is a sound algorithm for computing entailment.
- T is a complete algorithm for computing entailment.

Decision Procedures for Propositional Logic

- **Truth tables** provide a sound and complete decision procedure for testing satisfiability, validity, and entailment in propositional logic.
 - The proof is based on the observation that truth tables enumerate all possible models.
- Satisfiability, validity, and entailment in propositional logic are thus *decidable* problems.
- For problems involving a large number of atomic propositions the amount of calculation required by using truth tables may be prohibitive (always 2^n , where n is the number of atomic proposition involved in the formulas).

Reduction to Satisfiability

- A formula ϕ is satisfiable iff there is some interpretation \mathcal{I} (i.e., a truth value assignment) that satisfies ϕ (i.e., ϕ is true under \mathcal{I} , $\mathcal{I} \models \phi$, and \mathcal{I} is a model of ϕ).
- Validity, equivalence, and entailment can be reduced to satisfiability:
 - ϕ is a valid (i.e., a tautology) iff $\neg\phi$ is unsatisfiable.
 - ϕ entails ψ ($\phi \models \psi$) iff $\phi \rightarrow \psi$ is valid (*deduction theorem*).
 - $\phi \models \psi$ iff $\phi \wedge \neg\psi$ is unsatisfiable.
 - ϕ is equivalent to ψ ($\phi \equiv \psi$) iff $\phi \leftrightarrow \psi$ is valid.
 - $\phi \equiv \psi$ iff $\phi \models \psi$ and $\psi \models \phi$
- A *sound and complete* procedure deciding satisfiability is all we need: the *tableaux method* is a decision procedure which checks the existence of a model.

- The **Tableaux Calculus** is a decision procedure solving the problem of satisfiability.
- If a formula is satisfiable, the procedure will constructively exhibit a model of the formula.
- The basic idea is to incrementally build the model by looking at the formula, by **decomposing** it guided by its syntactic structure into **sets of literals**.
- The procedure exhaustively looks at all the possibilities, so that it can eventually prove that no model could be found for unsatisfiable formulas.

Negation Normal Form

The tableaux calculus works only if the formula has been translated into **Negation Normal Form** (NNF), i.e., all the negations have been pushed down using De Morgan laws.

Example::

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

becomes

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

The Tableaux Calculus builds a Tree where:

- The initial formula labels the root of the Tree;
- Each internal node has one or two child nodes depending on how a formula labeling the node is decomposed;
- The leaves are labeled by sets of literals;
- Each path in the Tree represents a possible interpretation for the formula;
- To build successor nodes in the Tree, so called **completion rules** are applied;
- The construction terminates when no more rules can be applied.

Tableaux Calculus: Completion Rules

To build successor nodes in the Tree, so called **completion rules** are applied.

$$\frac{\phi \wedge \psi}{\begin{array}{l} \phi \\ \psi \end{array}}$$

AND-rule. If a model satisfies a conjunction, then it also satisfies *each of* the conjuncts. This rule is **deterministic** and generates one successor node with both ϕ and ψ .

$$\frac{\phi \vee \psi}{\begin{array}{l} \phi \mid \psi \end{array}}$$

OR-rule. If a model satisfies a disjunction, then it also satisfies *one of* the disjuncts. It is a **non-deterministic** rule, and it generates two *alternative* branches of the tableaux.

Definition. For any atom p , the set $\{p, \neg p\}$ is a **complementary pair of literals** (called also a **clash**).

Theorem. A set of literals is satisfiable if and only if it does not contain a complementary pair of literals.

Proof Sketch. Let L be a set of literals. To show this theorem consider the following interpretation:

$$\forall p \in \Sigma,$$

$$\mathcal{I}(p) = \text{T} \quad \text{if } p \in L$$

$$\mathcal{I}(p) = \text{F} \quad \text{if } \neg p \in L$$

Simple examples (I)

$KB =$
 $\{ManUn \wedge ManCity, \neg ManUn\}$

$ManUn \wedge ManCity$
 $\neg ManUn$

$ManUn$
 $ManCity$



Simple examples (I)

$KB =$
 $\{ManUn \wedge ManCity, \neg ManUn\}$

$ManUn \wedge ManCity$
 $\neg ManUn \clubsuit$

$ManUn \clubsuit$
 $ManCity$



Simple examples (I)

$KB =$
 $\{ManUn \wedge ManCity, \neg ManUn\}$

$ManUn \wedge ManCity$
 $\neg ManUn$ ♣

$ManUn$ ♣
 $ManCity$ ♣

clash!

Simple examples (I)

$KB =$
 $\{ManUn \wedge ManCity, \neg ManUn\}$

$ManUn \wedge ManCity$
 $\neg ManUn$ ♣

$ManUn$ ♣
 $ManCity$ ♣

clash!

$KB =$
 $\{Chelsea \wedge ManCity, \neg ManUn\}$

$Chelsea \wedge ManCity$
 $\neg ManUn$

$Chelsea$
 $ManCity$

Simple examples (I)

$KB =$
 $\{ManUn \wedge ManCity, \neg ManUn\}$



$ManUn \wedge ManCity$
 $\neg ManUn$ 

$ManUn$ 
 $ManCity$ 

clash!

$KB =$
 $\{Chelsea \wedge ManCity, \neg ManUn\}$

$Chelsea \wedge ManCity$
 $\neg ManUn$ 

$Chelsea$ 
 $ManCity$ 

Simple examples (I)

$KB =$
 $\{ManUn \wedge ManCity, \neg ManUn\}$

$ManUn \wedge ManCity$
 $\neg ManUn$ ♣

$ManUn$ ♣
 $ManCity$ ♣

clash!

$KB =$
 $\{Chelsea \wedge ManCity, \neg ManUn\}$

$Chelsea \wedge ManCity$
 $\neg ManUn$ ♠

$Chelsea$ ♠
 $ManCity$ ♠

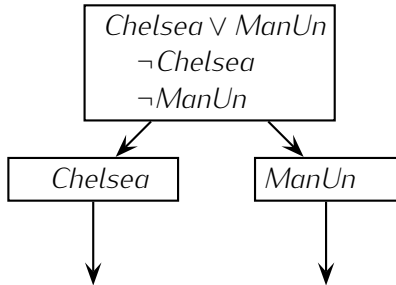
completed

Simple examples (II)

$$KB = \{Chelsea \vee ManUn, \\ \neg Chelsea, \neg ManUn\}$$

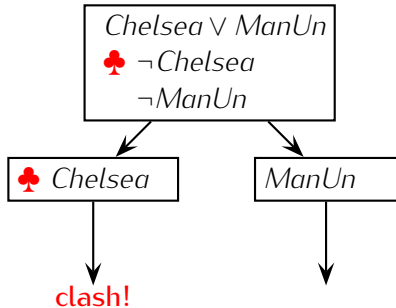
Simple examples (II)

$KB = \{Chelsea \vee ManUn, \neg Chelsea, \neg ManUn\}$



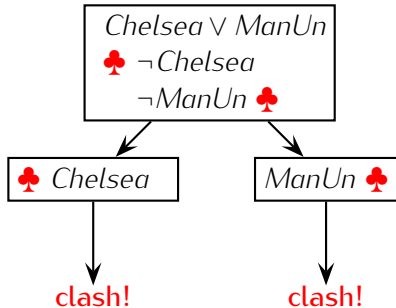
Simple examples (II)

$KB = \{Chelsea \vee ManUn, \neg Chelsea, \neg ManUn\}$

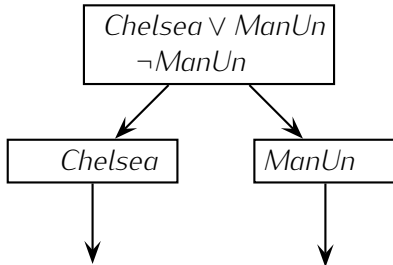


Simple examples (II)

$KB = \{Chelsea \vee ManUn, \neg Chelsea, \neg ManUn\}$

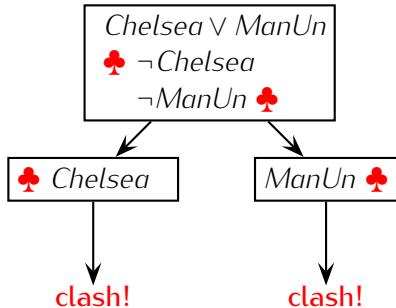


$KB = \{Chelsea \vee ManUn, \neg ManUn\}$

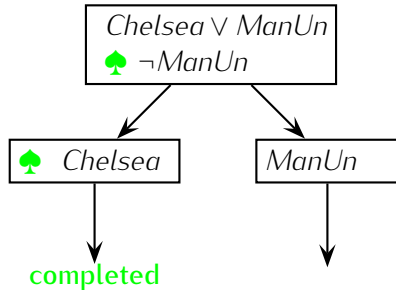


Simple examples (II)

$KB = \{Chelsea \vee ManUn, \neg Chelsea, \neg ManUn\}$

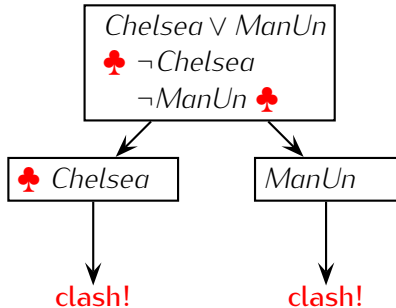


$KB = \{Chelsea \vee ManUn, \neg ManUn\}$

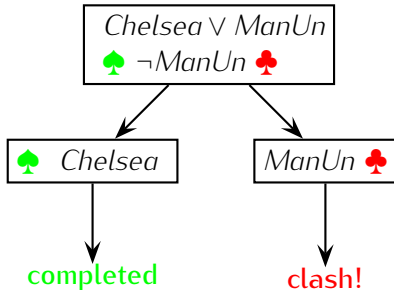


Simple examples (II)

$KB = \{Chelsea \vee ManUn, \neg Chelsea, \neg ManUn\}$



$KB = \{Chelsea \vee ManUn, \neg ManUn\}$



Instead of a branch with a **clash** and of a **completed** branch the following alternative definition is also used:

Definition

Given a Tableau for a formula φ then:

- A branch is said **closed** if it contains a clash; viceversa
- if no more rules are applicable then a branch is said to be **open**.

Tableaux Calculus

Finds a model for a given collection of sentences KB in negation normal form.

- 1 The **Tableaux Calculus** considers the knowledge base, KB , as the root node of a *refutation tree*.
- 2 Starting from the root, add new formulas to the tableaux, applying the *completion rules*.
- 3 Completion rules are either **deterministic**—they yield a uniquely determined successor node – or **nondeterministic**—yielding several possible alternative successor nodes (*branches*).
- 4 Apply the completion rules to each branch until either
(a) an explicit contradiction due to the presence of two complementary literals, $\{p, \neg p\}$, in a branch (a *clash*) is generated (i.e., an *closed* branch), or
(b) there is a *completed* branch where no more rules are applicable (i.e., an *open* branch).

Theorem. The construction of a tableau for any formula ϕ terminates.

Termination can be proved considering that:

- Rules are applied only once to each sub-formula;
- At each step the algorithm decomposes a formula into one or two *simpler* formulas, i.e., rules generate strict subformulas.

The Tableaux Calculus for Propositional Logic is Correct.

Theorem—Soundness and completeness. A formula ϕ is satisfiable if and only if the tableaux for ϕ contains a completed branch.

Soundness: If the tableaux has one completed branch then the formula is satisfiable.

To prove Soundness we need to show that the assignment that satisfies the set of literals labeling a completed branch can be extended to a model of the formula labeling the root.

There are four steps in the proof:

- 1 Define a property of formulas;
- 2 Show that the set of formulas in a completed branch has this property;
- 3 Prove that a set of formulas with this property is satisfiable;
- 4 Note that the formula in the root is in the set.

Step 1. Define a property of formulas.

Definition–Hintikka Set. Let Θ be a set of formulas. Then Θ is a **Hintikka set** iff:

- 1 For all atoms p appearing in a formula of Θ , either $p \in \Theta$ or $\neg p \in \Theta$.
- 2 If $\phi \wedge \psi \in \Theta$, then $\phi \in \Theta$ and $\psi \in \Theta$.
- 3 If $\phi \vee \psi \in \Theta$, then $\phi \in \Theta$ or $\psi \in \Theta$.

Step 2.

Lemma 1. If Θ is the union of formulas gathered on a path from the root to a completed leaf (i.e., a branch), then Θ is a Hintikka set.

Proof.

- Since the branch is completed, no complementary pair of literals appears in Θ , so Condition (1) holds for Θ .
- Conditions (2) and (3) easily hold since the branch is supposed to be completed, i.e., the completion rules have been applied and no more rules can be applied.

Step 3.

Lemma 2. Let Θ be a Hintikka set. Then Θ is satisfiable.

- We define an interpretation, \mathcal{I} , and then show that the interpretation is a model of Θ .

$$\begin{array}{ll} \mathcal{I}(p) = \text{T} & \text{if } p \in \Theta \\ \mathcal{I}(p) = \text{F} & \text{if } \neg p \in \Theta \\ \mathcal{I}(p) = \text{T} & \text{if } p \notin \Theta \text{ and } \neg p \notin \Theta \end{array}$$

By condition (1), \mathcal{I} is well defined.

- By structural induction we can easily show that $\forall \phi \in \Theta, \mathcal{I} \models \phi$.

Step 4.

Proof of Soundness.

- Assume that ϕ has a tableaux with a completed branch.
- By Lemma 1, Θ , the union of formulas on the nodes of that branch, is a Hintikka set.
- By Lemma 2, we can find an interpretation \mathcal{I} for Θ .
- Since $\phi \in \Theta$, then $\mathcal{I} \models \phi$.

Completeness: It is easier to prove the contrapositive: If the tableaux \mathcal{T} has a clash in every branch, then the formula is unsatisfiable.

Proof. We will prove a more general theorem:

if \mathcal{T}_n , the subtree rooted at node n of \mathcal{T} , closes then the set of formulas $\Theta(n)$ labeling n is unsatisfiable.

By induction on the height h_n on the node n of the Tree \mathcal{T}_n generated by the Completion Rules.

- **Base Case.** $h = 0$. Thus n is a leaf. Clearly a leaf is a set of literals and if it contains a clash it is unsatisfiable.
- **Induction Step.** By Inductive Hypothesis, for any node m root of a tree with height $h_m < h_n$, the set of formulas, Θ_m , labeling node m is unsatisfiable if the Tree rooted at m , say \mathcal{T}_m , contains a clash in every branch.

Tableaux Calculus: Completeness (cont.)

- **Case 1.** The determinist rule is applied to node n with height h_n . Then, $\Theta_n = \{\phi \wedge \psi\} \cup \Theta'_n$. Then, $\Theta_{n-1} = \{\phi, \psi\} \cup \Theta'_n$, with $h_{n-1} < h_n$. Now, if \mathcal{T}_n has a clash in every branch so is \mathcal{T}_{n-1} and by induction Θ_{n-1} is unsatisfiable, i.e., for all interpretation \mathcal{I} , either $\mathcal{I} \not\models \Theta'_n$, or $\mathcal{I} \not\models \{\phi, \psi\}$. In both cases, \mathcal{I} cannot be a model of Θ_n which is then unsatisfiable.
- **Case 2.** The non-determinist rule is applied to node n with height h_n . Then, $\Theta_n = \{\phi \vee \psi\} \cup \Theta'_n$. Then, $\Theta_{(n-1)} = \{\phi\} \cup \Theta'_n$, and $\Theta_{(n-1)'} = \{\psi\} \cup \Theta'_n$, with $h_{n-1} < h_n$. Now, if \mathcal{T}_n has a clash in every branch so are the two sub-Tree $\mathcal{T}_{(n-1)}$ and $\mathcal{T}_{(n-1)'}$ and by induction $\Theta_{(n-1)}$ and $\Theta_{(n-1)'}$ are both unsatisfiable. Thus, for all interpretation \mathcal{I} , either $\mathcal{I} \not\models \Theta'_n$ or both $\mathcal{I} \not\models \phi$ and $\mathcal{I} \not\models \psi$. In both cases, \mathcal{I} cannot be a model of Θ_n which is then unsatisfiable.

Tableaux Calculus Vs. Models: Summary

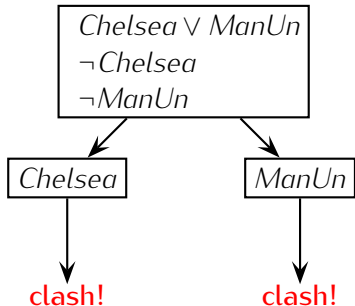
- A completed branch of the Tableaux gives a model of the *KB*: the *KB* is satisfiable. Since all formulas have been reduced (in the leaf) to sets of literals, it is possible to find an interpretation which make all the sentences in the branch true.
- If there is no completed branch (i.e., every branch has a clash), then it is not possible to find an assignment making the original *KB* true: the *KB* is unsatisfiable. In fact, the original formulas from which the tree is constructed can not be true simultaneously.

Entailment and Refutation

$\phi \models \psi$ iff $\phi \wedge \neg\psi$ is not satisfiable.

The tableaux may exhibit a counter-example (why?).

$\{Chelsea \vee ManUn, \neg ManUn\} \models$
Chelsea
(true)



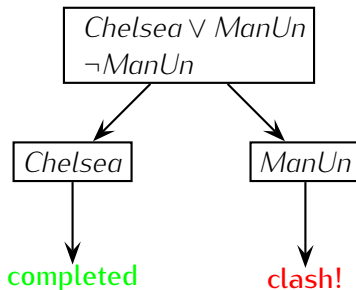
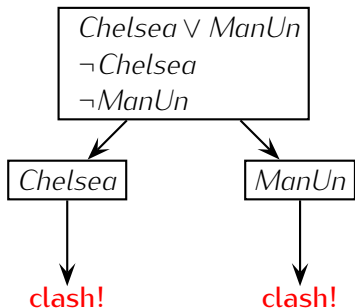
Entailment and Refutation

$\phi \models \psi$ iff $\phi \wedge \neg\psi$ is not satisfiable.

The tableaux may exhibit a counter-example (why?).

$\{Chelsea \vee ManUn, \neg ManUn\} \models$
 $Chelsea$
(true)

$\{Chelsea \vee ManUn\} \models ManUn$
(false)



The algorithm for constructing a tableau is not deterministic: at each steps, we need to choose:

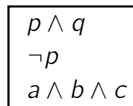
- 1 which leaf to extend, and
- 2 if the leaf contains more than one formula which is not a literal, which formula to decompose.

Efficiency: Heuristics

- 1 It is better to apply first AND-rules before OR-rules to avoid duplication: eg., try with $\phi = (p \vee q) \wedge (\neg p \wedge \neg q)$.
- 2 Closing a branch if it contains a formula and its negation, e.g., $(p \wedge (q, \vee r))$ Vs. $\neg(p \wedge (q, \vee r))$.
- 3 Be guided by the syntax looking for conflicts, e.g., try with $KB = p \wedge q, \neg p, a \wedge b \wedge c$

Efficiency: Heuristics

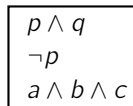
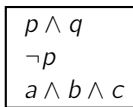
- 1 It is better to apply first AND-rules before OR-rules to avoid duplication: eg., try with $\phi = (p \vee q) \wedge (\neg p \wedge \neg q)$.
- 2 Closing a branch if it contains a formula and its negation, e.g., $(p \wedge (q, \vee r))$ Vs. $\neg(p \wedge (q, \vee r))$.
- 3 Be guided by the syntax looking for conflicts, e.g., try with $KB = p \wedge q, \neg p, a \wedge b \wedge c$



clash!

Efficiency: Heuristics

- 1 It is better to apply first AND-rules before OR-rules to avoid duplication: eg., try with $\phi = (p \vee q) \wedge (\neg p \wedge \neg q)$.
- 2 Closing a branch if it contains a formula and its negation, e.g., $(p \wedge (q, \vee r)) \vee \neg(p \wedge (q, \vee r))$.
- 3 Be guided by the syntax looking for conflicts, e.g., try with $KB = p \wedge q, \neg p, a \wedge b \wedge c$



Efficiency: comparison with truth tables

- The complexity of truth tables depends on the number of atomic formulas appearing in the KB ,
- the complexity of tableaux depends on the syntactic structure of the formulas in KB .

Try:

$$KB = ((p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee r) \wedge (\neg p \vee \neg r))$$

Tableaux as a Decision Procedure

Tableaux is a decision procedure for computing satisfiability, validity, and entailment in propositional logics:

- it is a sound algorithm
- it is a complete algorithm
- it is a terminating algorithm

Propositional Logic at work: the Graph Colouring Problem

The Graph Colouring problem is a well-known combinatorial problem from graph theory:

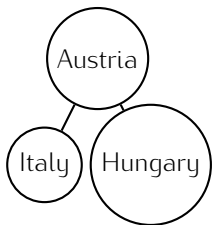
- A graph is defined as $G = (V, E)$, where $V = \{v_1, v_2, \dots, v_n\}$ is the set of *vertices* and $E = \{(v_i, v_j), \dots, (v_k, v_l)\}$ the set of *edges* connecting pairs of vertices.
- Find a colouring function $C : V \rightarrow \mathcal{N}$, such that connected vertices always have different colours.
- There is a *decision* variant of this problem: the question is to **decide whether for a particular number of colours, a coloring of the given graph exists.**

Encoding as satisfiability problem

A straightforward strategy for encoding the **Graph Colouring Decision Problem** into a satisfiability problem in propositional logic:

- Each assignment of a colour to a single vertex is represented by a propositional variable;
- Each colouring constraint (edge of the graph) is represented by a set of clauses ensuring that the **adjacent vertices** have different colours,
- Two additional sets of clauses ensure that valid assignments assign exactly one colour to each vertex.

Example



2 colors (Black and White): **Satisfiable**

Edge axioms:

$$(B_I \rightarrow \neg B_A) \wedge (W_I \rightarrow \neg W_A)$$

$$(B_H \rightarrow \neg B_A) \wedge (W_H \rightarrow \neg W_A)$$

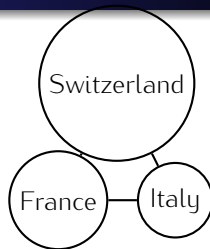
Node axioms:

$$(B_I \vee W_I), \dots$$

$$(B_I \rightarrow \neg W_I)$$

$$(B_H \rightarrow \neg W_H)$$

$$(B_A \rightarrow \neg W_A)$$



2 colors (Black and White): **Unsatisfiable**

Edge axioms:

$$(B_F \rightarrow \neg B_S) \wedge (W_F \rightarrow \neg W_S)$$

$$(B_S \rightarrow \neg B_I) \wedge (W_S \rightarrow \neg W_I)$$

$$(B_F \rightarrow \neg B_I) \wedge (W_F \rightarrow \neg W_I)$$

Node axioms:

$$(B_F \vee W_F), \dots$$

$$(B_F \rightarrow \neg W_F)$$

$$(B_S \rightarrow \neg W_S)$$

$$(B_I \rightarrow \neg W_I)$$

Complexity of the problem

<i>vertices</i>	<i>edges</i>	<i>colours</i>	<i>vars</i>	<i>clauses</i>
30	60	3	90	300
50	115	3	150	545
75	180	3	225	840
100	239	3	300	1117
125	301	3	375	1403
150	360	3	450	1680
175	417	3	525	1951
200	479	3	600	2237

Other hot problems for propositional logic

- The general scenario in **Blocks World** planning comprises a number of blocks and a table. The blocks can be piled onto each other, where the downmost block of a pile is always on the table. Given an initial and a goal configuration of blocks, the problem is to find a sequence of single block move operations which, when applied to the initial configuration, leads to the goal situation.
- In the **Logistics** planning domain, packages have to be moved between different locations in different cities. Within cities, packages are carried by trucks while between cities they are flown in planes. Both, trucks and airplanes are of limited capacity.
- **Circuit fault** analysis, **Scheduling**, . . .