Logic

Deduction in Propositional Logic

Enrico Franconi

franconi@inf.unibz.it
http://www.inf.unibz.it/~franconi

Faculty of Computer Science, Free University of Bozen-Bolzano

Decision Procedures in Logic: soundness

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

$$\boxed{KB \vdash_i \alpha}$$

- 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:

$$\boxed{KB \vdash_i \alpha}$$

- 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

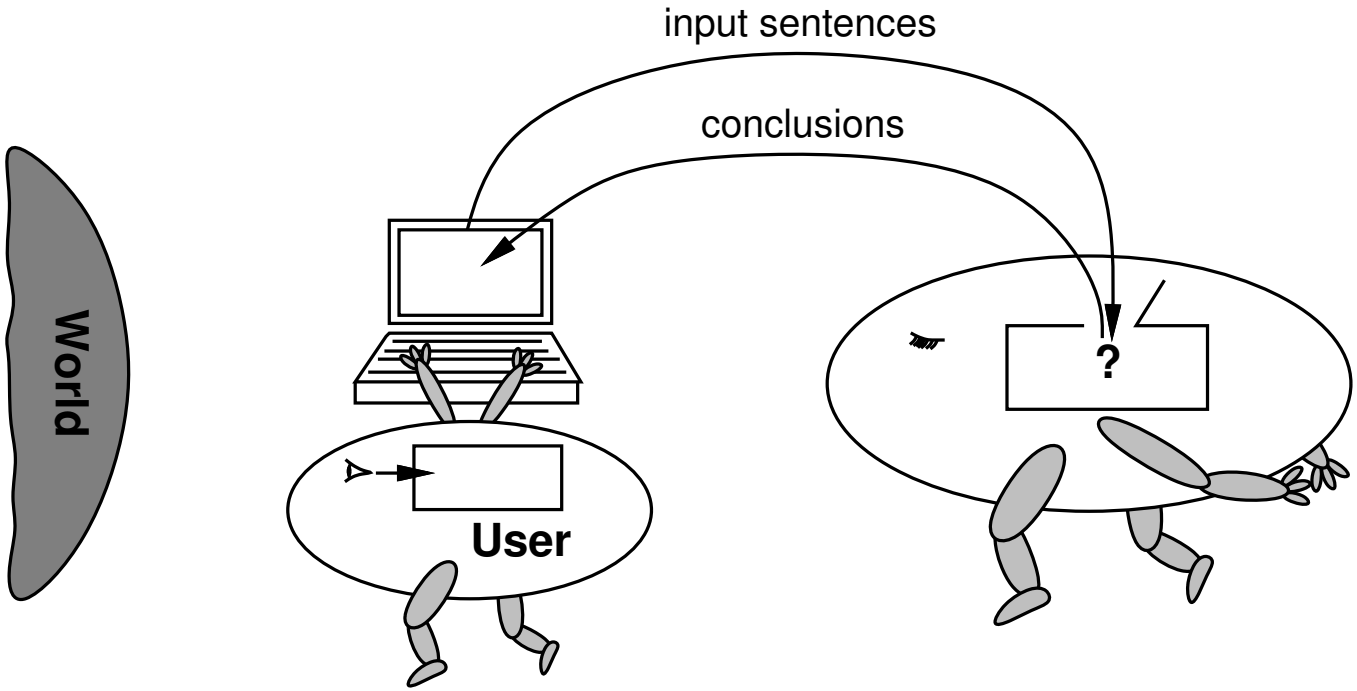
Let's consider the problem of computing entailment between formulas;

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

Dual problems

Can we use a sound but incomplete decision procedure for a problem to solve the **dual problem** by inverting the answers?

T is an unsound procedure for computing non-entailment between formulas.



Incompleteness of the reasoning procedures of the reasoning agent leads to *unsound* reasoning of the whole agent, if the main system relies on *negative conclusions* of the reasoning agent module.

Propositional Decision Procedures

- 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 $\mathcal{I}$: $\mathcal{I} \models \phi$).
- Validity, equivalence, and entailment can be reduced to satisfiability:
  - ϕ is a valid (i.e., a tautology) iff $\neg\phi$ is not satisfiable.
  - ϕ entails ψ ($\phi \models \psi$) iff $\phi \rightarrow \psi$ is valid (*deduction theorem*).
    - $\boxed{\phi \models \psi \text{ iff } \phi \wedge \neg\psi \text{ is not satisfiable}}$.
  - 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, and the *tableaux method* is a decision procedure which checks the existence of a model.

Tableaux Calculus

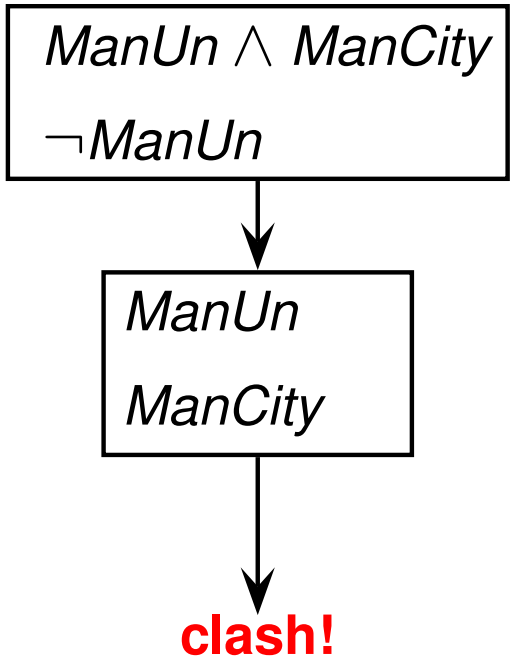
- The Tableaux Calculus is a decision procedure solving the problem of satisfiability.
- If a formula is satisfiable, the procedure will constructively exhibit a model of the formula.
- The basic idea is to incrementally build the model by looking at the formula, by decomposing it in a top/down fashion. The procedure exhaustively looks at all the possibilities, so that it can eventually prove that no model could be found for unsatisfiable formulas.

Simple examples (I)

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

Simple examples (I)

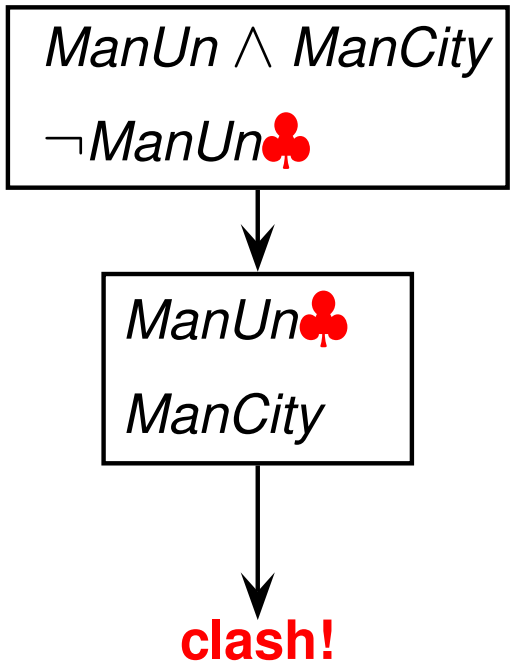
$KB = ManUn \wedge ManCity, \neg ManUn$



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

Simple examples (I)

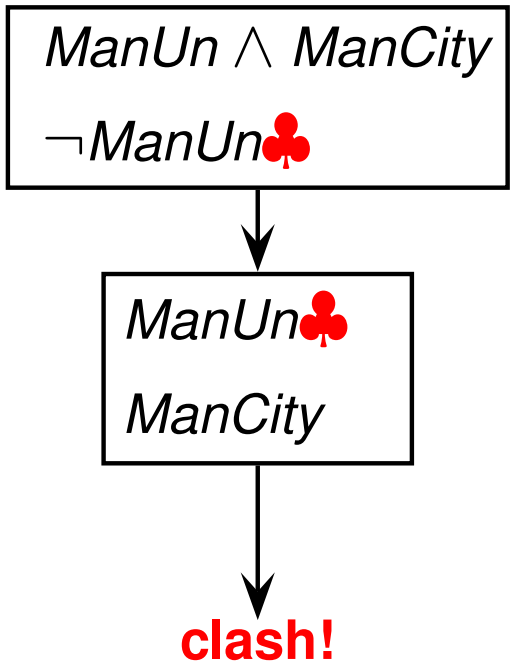
$KB = ManUn \wedge ManCity, \neg ManUn$



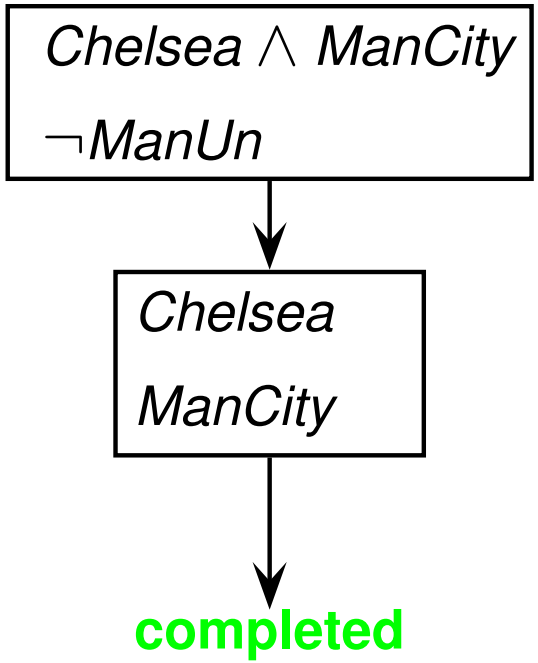
$KB = Chelsea \wedge ManCity, \neg ManUn$

Simple examples (I)

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

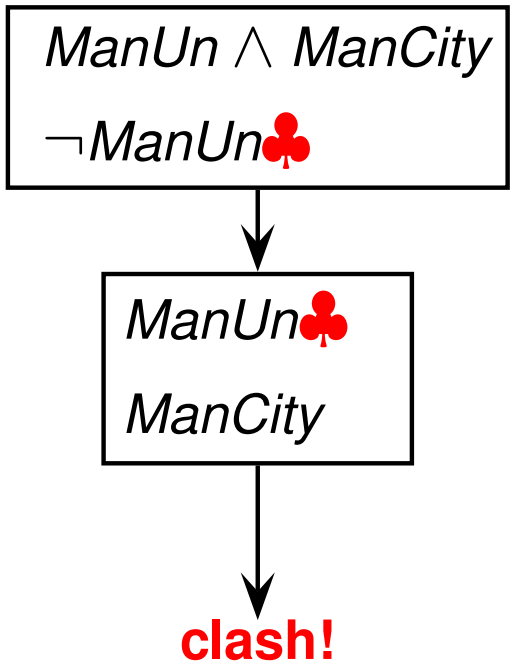


$KB = Chelsea \wedge ManCity, \neg ManUn$

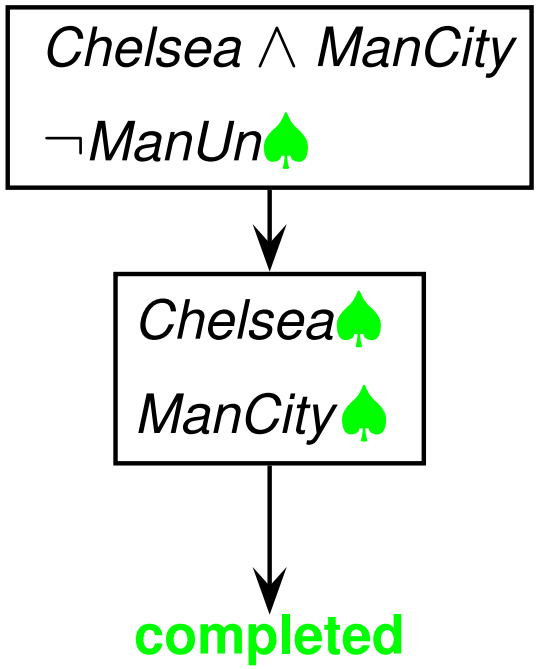


Simple examples (I)

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



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



Simple examples (II)

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

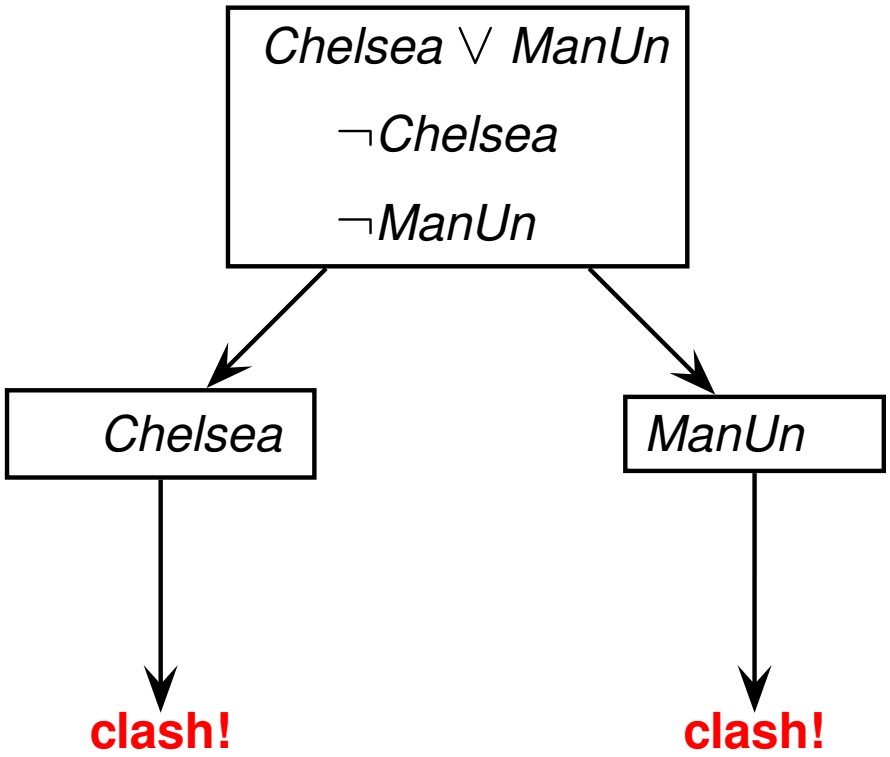
$$KB = Chelsea \vee ManUn, \neg Chelsea, \neg ManUn$$

Simple examples (II)

$KB =$

$Chelsea \vee ManUn, \neg Chelsea, \neg ManUn$

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

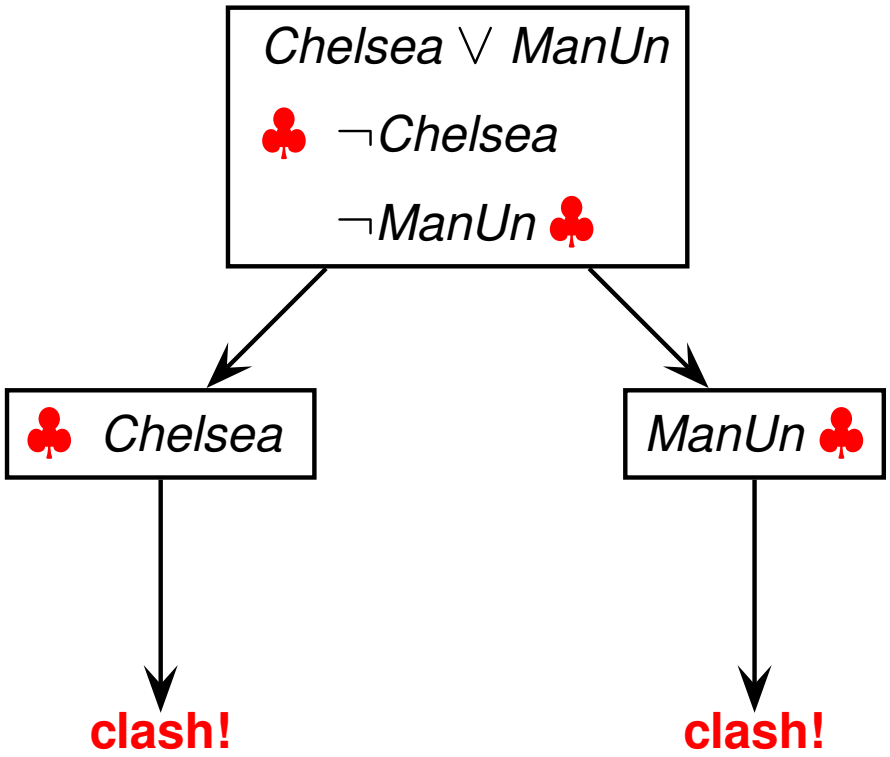


Simple examples (II)

$KB =$

$Chelsea \vee ManUn, \neg Chelsea, \neg ManUn$

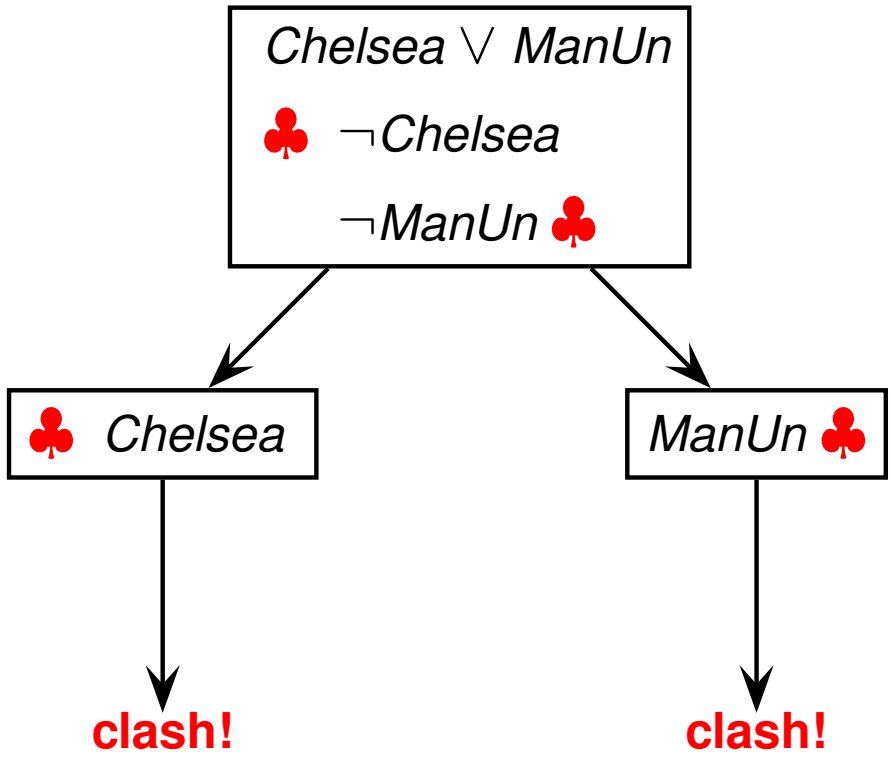
$KB = Chelsea \vee ManUn, \neg ManUn$



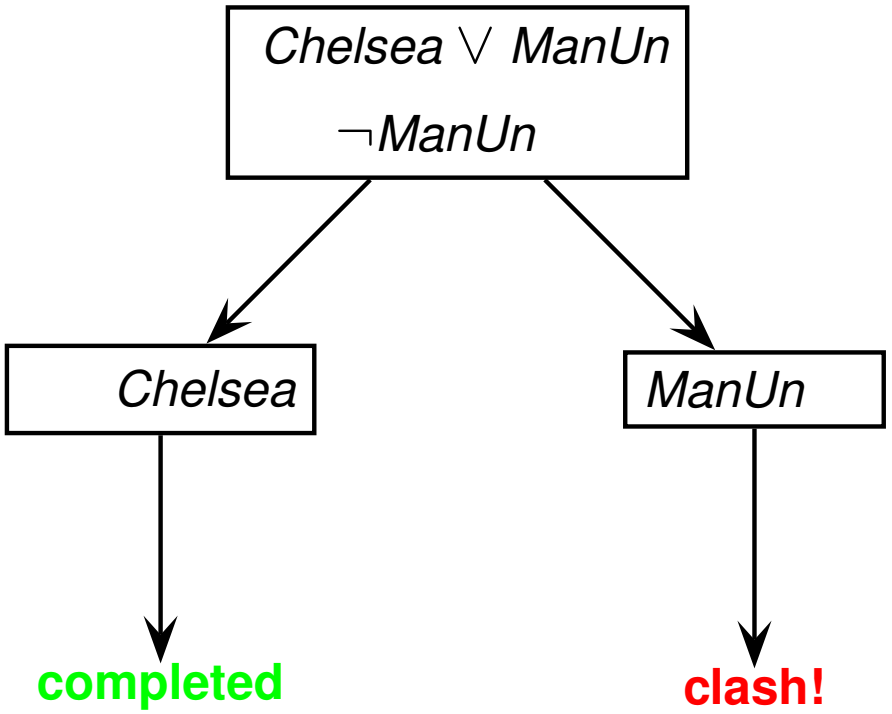
Simple examples (II)

$KB =$

$Chelsea \vee ManUn, \neg Chelsea, \neg ManUn$



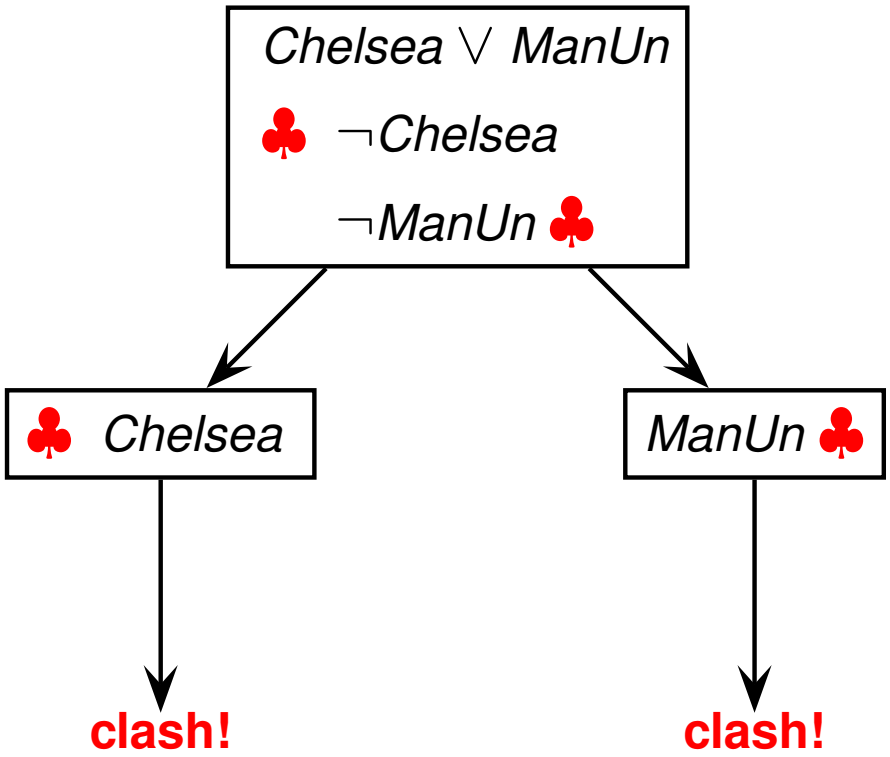
$KB = Chelsea \vee ManUn, \neg ManUn$



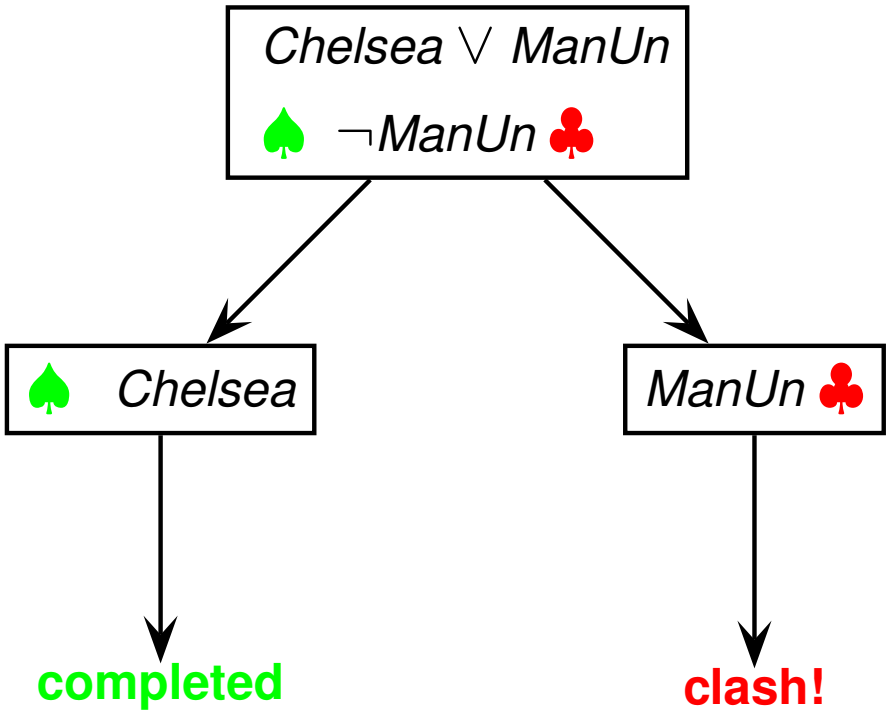
Simple examples (II)

$KB =$

$Chelsea \vee ManUn, \neg Chelsea, \neg ManUn$



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



Tableaux Calculus

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

1. Consider the knowledge base *KB* as the root node of a *refutation tree*. A node in a refutation tree is called *tableaux*.

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 until either
   (a) an explicit contradiction due to the presence of two opposite literals in a node (a *clash*) is generated in each branch, or
   (b) there is a *completed* branch where no more rule is applicable.

Models

- The completed branch of the refutation tree gives a model of *KB*: the *KB* is satisfiable. Since all formulas have been reduced to literals (i.e., either positive or negative atomic propositions), it is possible to find an assignment of truth and falsity to atomic sentences 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.

The Calculus

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

If a model satisfies a conjunction, then it also satisfies *each of* the conjuncts

$$\frac{\phi \vee \psi}{\phi \mid \psi}$$

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.

Negation Normal Form

The given tableaux calculus works only if the formula has been translated into Negation Normal Form, i.e., all the negations have been pushed down.

Example::

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

becomes

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

Entailment and Refutation

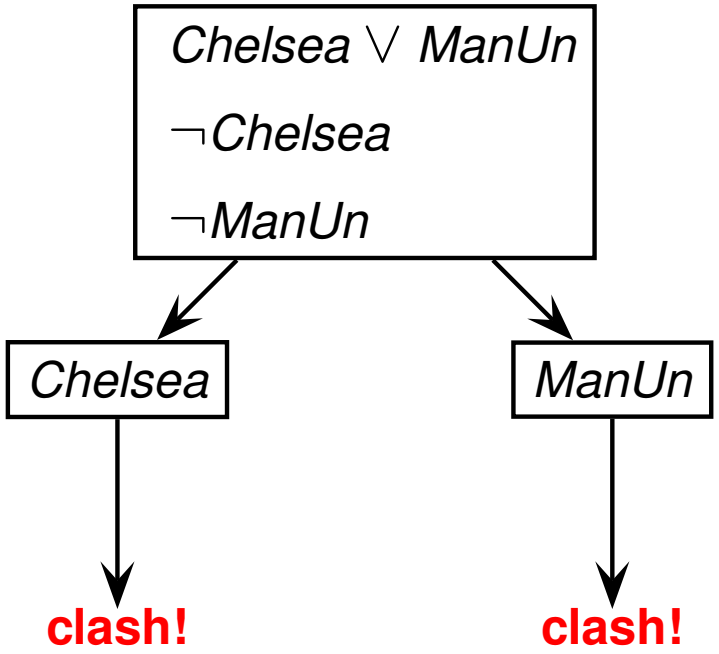
$\phi \models \psi$ iff $\phi \wedge \neg\psi$ is not satisfiable. The tableaux may exhibit a counter-example (why?).

$Chelsea \vee ManUn, \neg ManUn \models Chelsea$

(true)

$Chelsea \vee ManUn \models ManUn$

(false)

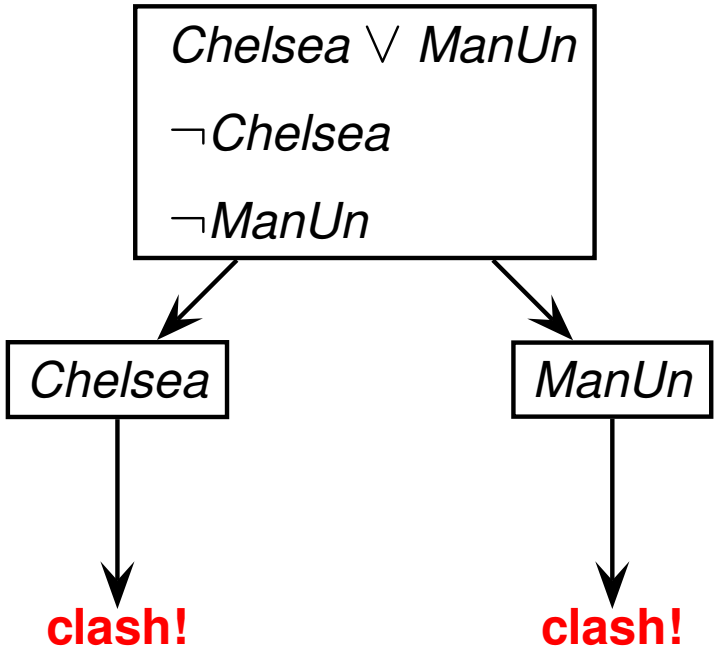


Entailment and Refutation

$\phi \models \psi$ iff $\phi \wedge \neg\psi$ is not satisfiable. The tableaux may exhibit a counter-example (why?).

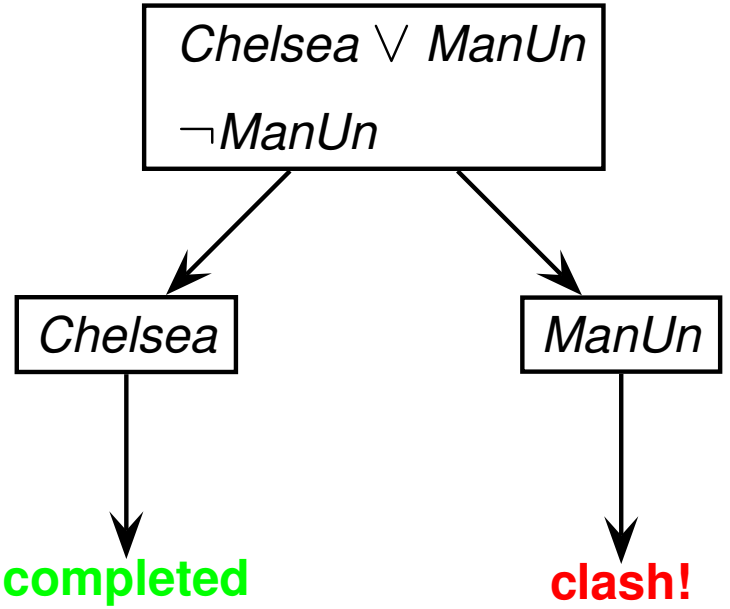
$Chelsea \vee ManUn, \neg ManUn \models Chelsea$

(true)



$Chelsea \vee ManUn \models ManUn$

(false)

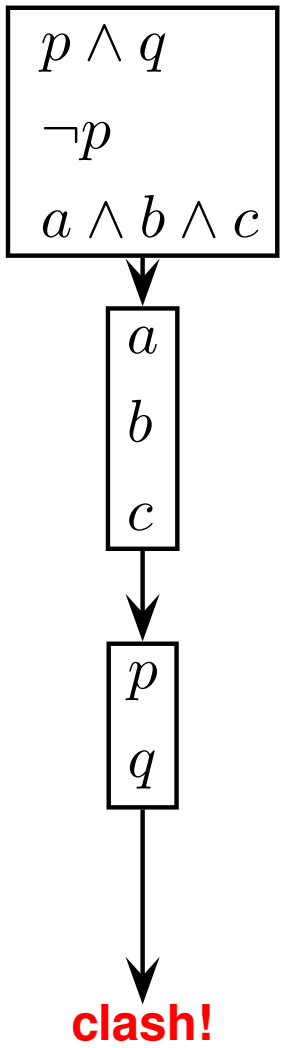


Efficiency: order of rule application

$KB = p \wedge q, \neg p, a \wedge b \wedge c$

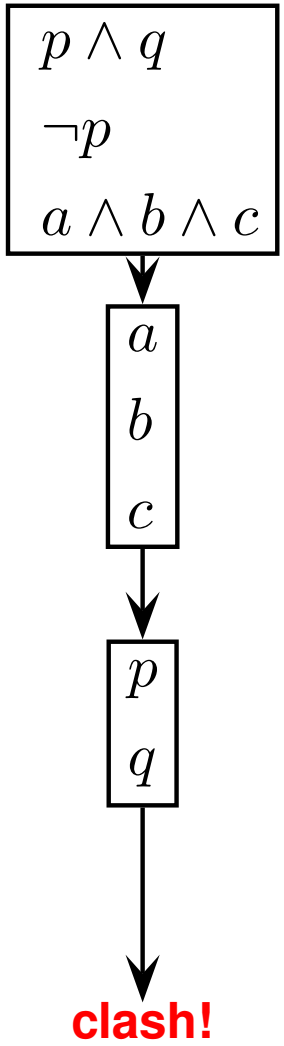
Efficiency: order of rule application

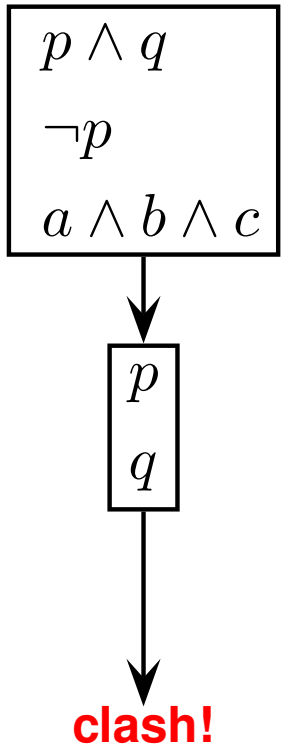
$KB = p \wedge q, \neg p, a \wedge b \wedge c$



Efficiency: order of rule application

$KB = p \wedge q, \neg p, a \wedge b \wedge c$





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