Tableaux

Jiří Vyskočil

2017

Tableau /tæbloʊ/ methods

- Tableau method is another useful deduction method for automated theorem proving in propositional, first-order, modal, temporal and many other logics.
- The Semantic tableau is tree whose nodes are labeled with formulae.
- The expansion rules transforms semantic tableau into one having an equivalent represented formula.
- The main principle of tableaux is to attempt to "break" complex formulae into smaller ones until complementary pairs of literals are produced or no further expansion is possible.

First-Order Logic Tableau

- An input for tableau is set of first-order formulae.
- The goal is to find contradiction in the input set of formulae (similar to resolution method).
- For simplicity let's assume, that we have only the following logical connectives:

  & (conjunction), $\vee$ (disjunction), and $\neg$ (negation).
- If we want to use more logical connectives, then we can define them using the previous ones (&, $\vee$, and $\neg$) and we can expand the new ones before starting tableau method.

First-Order Logic Tableau

- Initial tableau is only one node/root that contains conjunction of all input set formulae.
- Sometimes the root of tableau is labeled as **T** (top) only.
- Now we can start iteratively applying tableau expansion rules.

conjunction (&)

- If there is a node on a branch in tableau of the form:

  A & B,

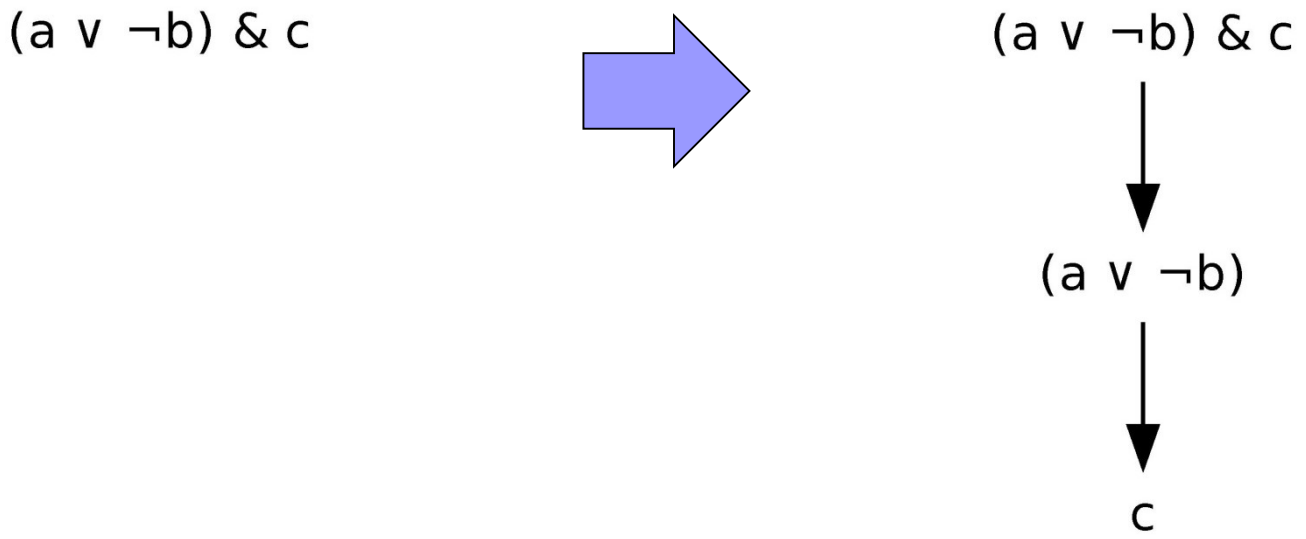
  where A and B are arbitrary formulae, we connect to a leaf of the branch a new branch with standalone formulae A and B as nodes.

- Formally:

$$(\&)\ \frac{A \,\&\, B}{\begin{array}{c} A \\ B \end{array}}$$

conjunction (&) – example

- The following example shows the initial and the resulting tableau after applying the conjunction rule.



disjunction ($\vee$)

- If there is a node on a branch in tableau of the form:

  $A \vee B$,

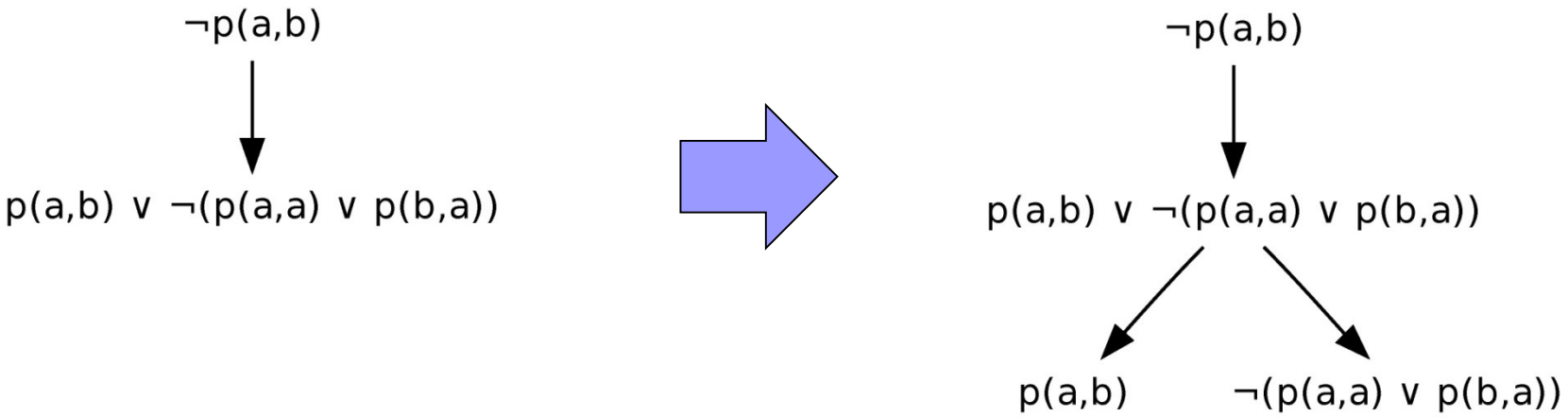
  where A and B are arbitrary formulae, we connect to the leaf of the branch two new branches. The first will contain formula A and the second formula B.

- Formally:

$$(\vee)\ \frac{A \vee B}{A \mid B}$$

disjunction ($\vee$) – example

- The following example shows the initial and the resulting tableau after applying the disjunction rule.



NNF (Negation Normal Form)

- A first-order formula is in *negated normal form* (NNF) if the negation operator ¬ is only applied to predicates.
- Every formula in first-order logic can be translated into NNF using:
  - De Morgan's laws:

    $\neg(A \vee B) \Leftrightarrow (\neg A \,\&\, \neg B)$,

    $\neg(A \,\&\, B) \Leftrightarrow (\neg A \vee \neg B)$,
  - quantifier negation rules:

    $\neg(\forall x\ \varphi) \Leftrightarrow (\exists x\ \neg\varphi)$,

    $\neg(\exists x\ \varphi) \Leftrightarrow (\forall x\ \neg\varphi)$
  - double negation rule:

    $(\neg\neg A \Leftrightarrow A)$.
- Translation of formulae into NNF is usually applied before tableau method.

negation ($\neg$)

- If the tableau method is applied on formula in NNF then we do not need any further rules for negation.
- If we do not have formula in NNF then we have to introduce the following rules for handling negation:

$$(\neg 1)\ \frac{\neg\neg A}{A}$$

$$(\neg 2)\ \frac{\neg(A \,\&\, B)}{\neg A \vee \neg B} \qquad (\neg 3)\ \frac{\neg(A \vee B)}{\neg A \,\&\, \neg B}$$

$$(\neg 4)\ \frac{\neg\,(\forall x\ \varphi)}{\exists x\ \neg\varphi} \qquad (\neg 5)\ \frac{\neg\,(\exists x\ \varphi)}{\forall x\ \neg\varphi}$$

universal quantification (∀)

- If there is a node on a branch in tableau of the form:

$$\forall x\ \varphi(x)$$

where x is a variable and $\varphi(x)$ is a formula that contains free variable x. We connect a new formula $\varphi(x')$ to a leaf of the branch, where $\varphi(x')$ can be obtained from formula $\varphi(x)$ by substitution of all free occurrences of x by a new fresh variable x' that does not occur anywhere in the whole current tableau.

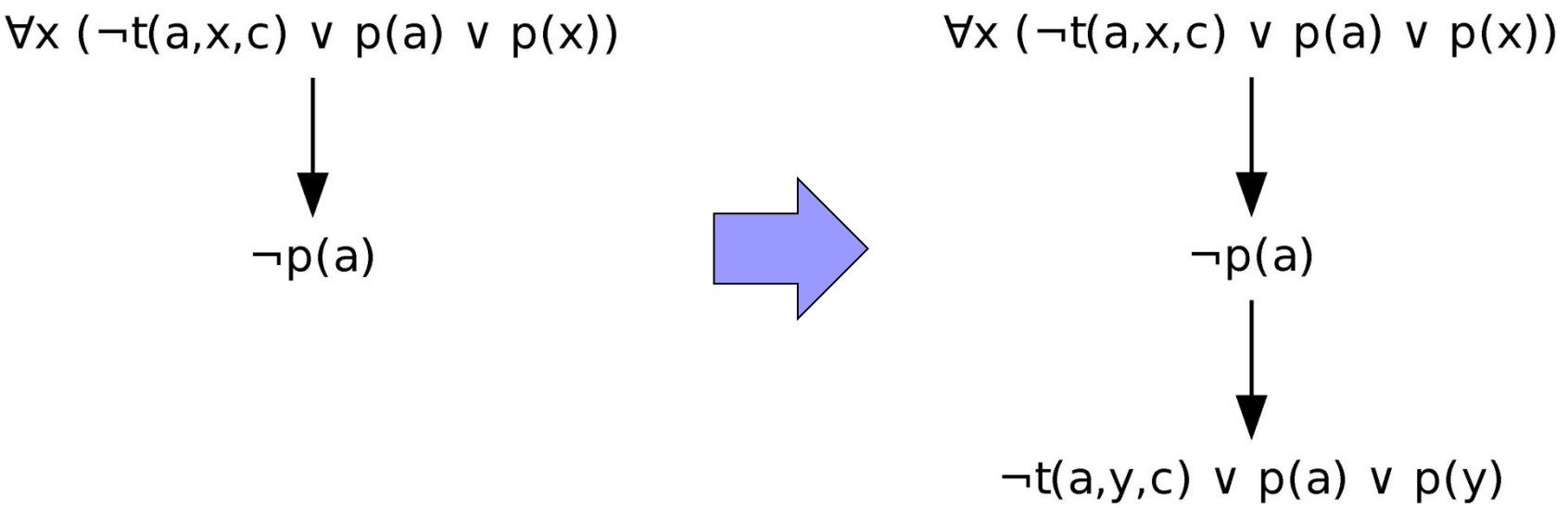
- Formally:

$$(\forall)\ \frac{\forall x\ \varphi(x)}{\varphi(x')}$$

where x' is a new fresh variable, that does not occur anywhere in the current tableau

universal quantification(∀) – example

- The following example shows the initial and the resulting tableau after applying the universal quantification rule.



existential quantification(∃)

- If there is a node on a branch in tableau of the form:

$$\exists x\ \delta(x)$$

where x is a free variable in formula $\delta(x)$ and $\delta(x)$ contains free variables $x_1,\ldots,x_n$ then we connect to the current leaf of the current branch a new formula node $\delta(f(x_1,\ldots,x_n))$, where $\delta(f(x_1,\ldots,x_n))$ can be obtained from $\delta(x)$ by substitution of all free occurrences of x by Skolem term $f(x_1,\ldots,x_n)$. Where f is a new function symbol in the whole current tableau.

- Formally:

$$(\exists)\ \frac{\exists x\ \delta(x)}{\delta(f(x_1,\ldots,x_n))}$$

where f is a new function symbol and $x_1,\ldots,x_n$ are free variables in formula δ

closed branch

- IF there are two complementary literals L and K on some branch B of the current tableau such that there exists MGU (most general unifier) θ of L and $\neg K$ (resp. $\neg L$ a K) that $L\,\theta \equiv \neg K\,\theta$ (resp. $\neg L\,\theta \equiv K\,\theta$),
- THEN apply θ on all nodes of the current tableau and label branch B as closed.
- Closed branches do not need any further expansion.
- Example:

  If $L = \neg p(x)$ and $K = p(t)$,

  then there exists a substitution $\theta = \{ x / t \}$ such that $\neg p(t) \equiv \neg p(t)$.

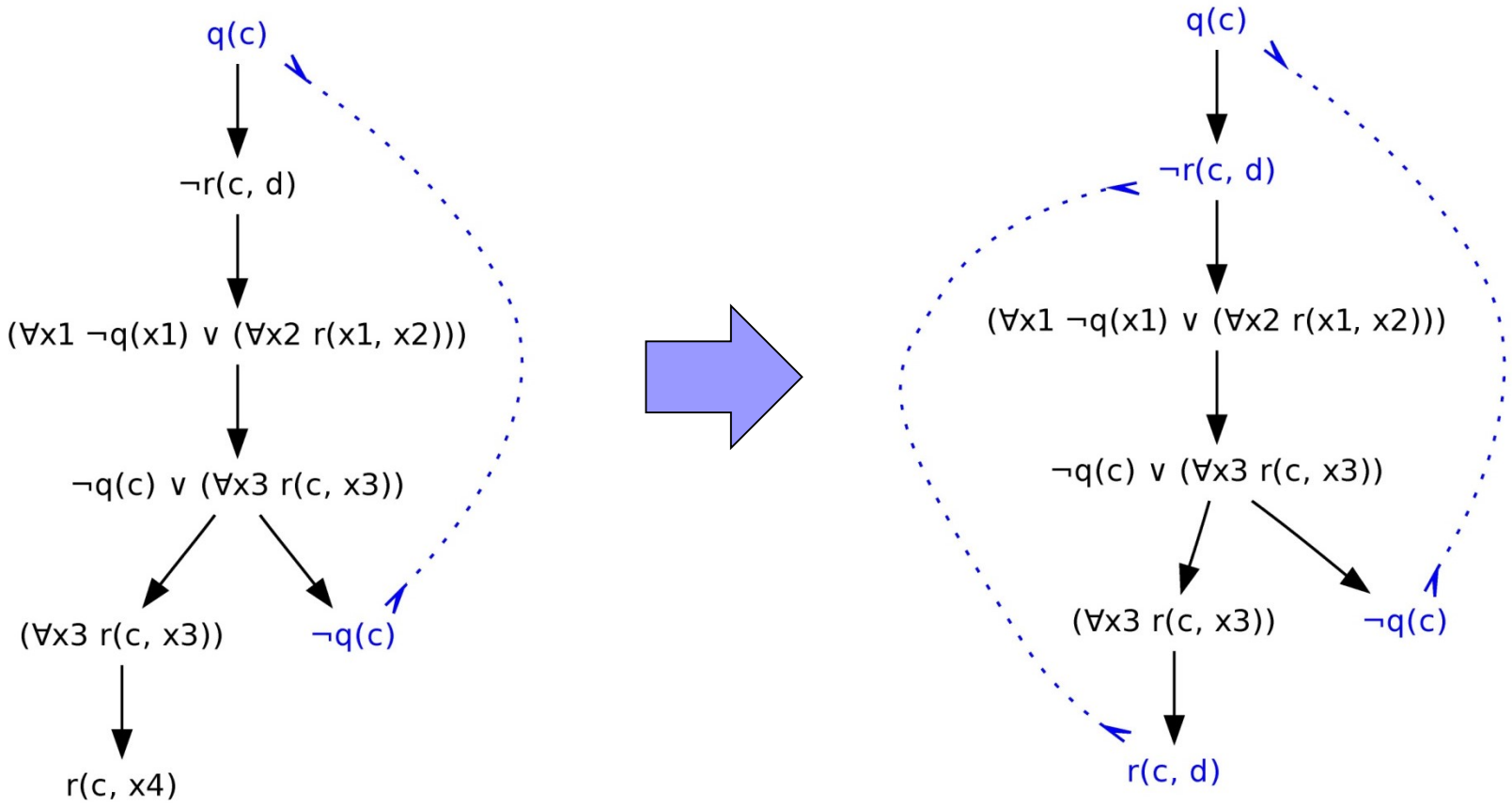
closed tableau

- **Tableau is closed**, if all of its branches are closed.

- Obtaining a tableau where all branches are closed is a way for proving the unsatisfiability of the original set. All tableau rules are logically correct. The root of tableau implies every node in tableau. Every path from root to leaf contains at least two complementary literals that imply contradiction.

closed tableau – example

- The following example shows the initial and the resulting tableau after closing the left branch (the right branch has been already closed).
- The resulting tableau is closed.



LeanTAP

- LeanTAP is one of the shortest complete first-order prover.
- LeanTAP is based on tableau method.
- LeanTAP is five clause Prolog program.
- An input is a conjunction of skolemized closed formulae in NNF.
- Formula representation:
  - logical connectives: `,` (conjunction), `;` (disjunction), `-` (negation)
  - universal quantifier: `all(`*variable*`,`*formula*`)`
  - variables are Prolog variables (starting with capital letter)
  - functions and predicates have Prolog notation
- LeanTAP's source code follows:

LeanTAP

```
% prove(+Fml,+UnExp,+Lits,+FreeV,+VarLim)

prove((A,B),UnExp,Lits,FreeV,VarLim) :- !,
        prove(A,[B|UnExp],Lits,FreeV,VarLim).

prove((A;B),UnExp,Lits,FreeV,VarLim) :- !,
        prove(A,UnExp,Lits,FreeV,VarLim),
        prove(B,UnExp,Lits,FreeV,VarLim).

prove(all(X,Fml),UnExp,Lits,FreeV,VarLim) :- !,
        \+ length(FreeV,VarLim),
        copy_term((X,Fml,FreeV),(X1,Fml1,FreeV)),
        append(UnExp,[all(X,Fml)],UnExp1),
        prove(Fml1,UnExp1,Lits,[X1|FreeV],VarLim).

prove(Lit,_,[L|Lits],_,_) :-
        (Lit = -Neg; -Lit = Neg) ->
           (unify_with_occurs_check(Neg,L)
         ; prove(Lit,[],Lits,_,_)).

prove(Lit,[Next|UnExp],Lits,FreeV,VarLim) :-
        prove(Next,UnExp,[Lit|Lits],FreeV,VarLim).
```

LeanTAP

an input formula

a queue of formulae to explore

a list of literals on a current branch of tableau

a list of all current free variables

```
% prove(+Fml,+UnExp,+Lits,+FreeV,+VarLim)

prove((A,B),UnExp,Lits,FreeV,VarLim) :- !,
        prove(A,[B|UnExp],Lits,FreeV,VarLim).

prove((A;B),UnExp,Lits,FreeV,VarLim) :- !,
        prove(A,UnExp,Lits,FreeV,VarLim),
        prove(B,UnExp,Lits,FreeV,VarLim).

prove(all(X,Fml),UnExp,Lits,FreeV,VarLim) :- !,
        \+ length(FreeV,VarLim),
        copy_term((X,Fml,FreeV),(X1,Fml1,FreeV)),
        append(UnExp,[all(X,Fml)],UnExp1),
        prove(Fml1,UnExp1,Lits,[X1|FreeV],VarLim).

prove(Lit,_,[L|Lits],_,_) :-
        (Lit = -Neg; -Lit = Neg) ->
           (unify_with_occurs_check(Neg,L)
         ; prove(Lit,[],Lits,_,_)).

prove(Lit,[Next|UnExp],Lits,FreeV,VarLim) :-
        prove(Next,UnExp,[Lit|Lits],FreeV,VarLim).
```

max. number of different free variables allowed in tableau. It determines max. depth of tableau.

conjunction case

disjunction case

universal quantifier case

literal case

If it is not possible to close the current branch of tableau then LeanTAP continues with processing of the unexplored formulae.

LeanTAP

- If you want to start the prover then you need to form the following Prolog query:

```
?- prove(<conjunction of skolemized formulae in NNF>,
         [],[],[],
         <max. number of free variables in tableau>).
```

Example

Does John like peanuts?

John likes all kind of foods.

Apples and chicken are food.

Bill is alive.

If someone is alive then

he/she has not been killed by anything.

Bill eats peanuts.

Sue eats everything that Bill eats.

Anything anyone eats and isn’t killed by is food.

Example - Formalization

Does John like peanuts?	likes(john,peanuts)
John likes all kind of foods.	$\forall X$ food(X) $\rightarrow$ likes(john,X)
Apples and chicken are food.	food(apples) & food(chicken)
Bill is alive.	alive(bill)
If someone is alive then	
he/she has not been killed by anything.	$\forall Y$ alive(Y) $\rightarrow$ $\forall X$ not_killed_by(Y,X)
Bill eats peanuts.	eats(bill,peanuts)
Sue eats everything that Bill eats.	$\forall X$ eats(bill,X) $\rightarrow$ eats(sue,X)

Anything anyone eats and isn't killed by is food.

$$\forall X \ (\exists Y \ \text{eats}(Y,X) \ \& \ \text{not\_killed\_by}(Y,X)) \rightarrow \text{food}(X)$$

Example – Negated Conjunction + Translation to NNF

likes(john,peanuts)

```
-likes(john, peanuts),
```

$\forall X$ food(X) $\rightarrow$ likes(john,X)

```
all(X,(-food(X);likes(john, X)),
```

food(apples) & food(chicken)

```
food(apples), food(chicken),
```

alive(bill)

```
alive(bill),
```

$\forall Y$ alive(Y) $\rightarrow$ $\forall X$ not_killed_by(Y,X)

```
all(Y,(-alive(Y);all(X,not_killed_by(Y, X)))),
```

eats(bill,peanuts)

```
eats(bill, peanuts),
```

$\forall X$ eats(bill,X) $\rightarrow$ eats(sue,X)

```
all(X,(-eats(bill, X);eats(sue, X))),
```

$\forall X$ ($\exists Y$ eats(Y,X) & not_killed_by(Y,X)) $\rightarrow$ food(X)

```
all(X,(food(X);all(Y,(-eats(Y,X);-not_killed_by(Y,X))))).
```

Example – LeanTAP call

```
?- prove((-likes(john, peanuts),
        food(apples),
        food(chicken),
        all(X,(food(X);all(Y,(-eats(Y,X);-not_killed_by(Y,X))))),
        alive(bill),
        eats(bill, peanuts),
        all(X,(-eats(bill, X);eats(sue, X))),
        all(Y,(-alive(Y);all(X,not_killed_by(Y, X)))),
        all(X,(-food(X);likes(john, X)))),
        [],
        [],
        [],
        5).
```

Example – Tableau Proof