Logical reasoning and programming

First-order tableaux and model finding

Karel Chvalovský

CIIRC CTU

Section 1

Tableaux

Free-variable tableaux

We again want to solve the problem

$$\Gamma \models \varphi$$

where Γ is a finite set of formulae and φ is a formula. This problem is equivalent to

$$\models \bigwedge \Gamma \to \varphi$$

and hence

$$\bigwedge \Gamma \wedge \neg\varphi \text{ is unsatisfiable.}$$

We no longer require that they are in a CNF. On the contrary, we will have rules for different connectives and quantifiers.

Our approach will be simple—we will expand formulae hoping to show that they are trivially inconsistent, or no further expansion is possible.

Proof tree and branches

We construct a tree where in the root we have a formula (or a set of formulae) that we want to show is unsatisfiable.

$$\dfrac{p(a) \wedge p(b) \wedge (\neg p(a) \vee \neg p(b))}{p(a)}\,(\wedge)$$

$$p(b)$$

$$\dfrac{\neg p(a) \vee \neg p(b)}{\dfrac{\neg p(a)}{\times} \qquad \dfrac{\neg p(b)}{\times}}\,(\vee)$$

Branch

We say that a set of formulae on a path from the root to a leaf is a branch. Our goal is to show that a branch is trivially unsatisfiable.

Closed branch and closed tableau

We say that a branch is closed, if it contains complementary literals p and $\neg q$, where p and q are atomic predicates. We say that a tableau is closed, if all its branches are closed. It means that the formula (or formulae) we started with is unsatisfiable.

Conjunctive rule (α-rule)

Let a conjunction $\alpha_1 \wedge \cdots \wedge \alpha_n$ be on a branch that is a set of formulae Π. If Π is satisfiable, then $\Pi \cup \{\alpha_1, \ldots, \alpha_n\}$ is also satisfiable. Hence we can expand the conjunction and add all its conjuncts to the branch.

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n}{\begin{array}{c}\alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n\end{array}} (\wedge)$$

Disjunctive rule (β-rule)

Let a disjunction $\beta_1 \vee \cdots \vee \beta_n$ be on a branch that is a set of formulae Π. If Π is satisfiable, then at least one of the sets $\Pi \cup \{\beta_1\}, \ldots, \Pi \cup \{\beta_n\}$ is also satisfiable. Hence we can split our current branch into n branches and by showing that all of them are unsatisfiable, we show that Π is also unsatisfiable.

$$\frac{\beta_1 \vee \beta_2 \vee \cdots \vee \beta_n}{\beta_1 \quad \beta_2 \quad \cdots \quad \beta_n} \, (\vee)$$

Universal rule (γ-rule)

Let a formula $\forall X \gamma(X)$ be on a branch that is a set of formulae Π. If Π is satisfiable, then $\Pi \cup \{\gamma(t)\}$ for all terms t, where t is substituable into $\gamma(X)$, is also satisfiable. We can keep the choice of a particular t open as long as possible by introducing a fresh variable for it.

$$\frac{\forall X \gamma(X)}{\gamma(X')} \, (\forall)$$

where X' is a fresh variable not occurring in the tableau. $\gamma(X')$ means that all free occurrences of X in $\gamma(X)$ are replaced by X'.

Existential rule (δ-rule)

Let a formula $\exists X \delta(X)$ be on a branch that is a set of formulae Π. If Π is satisfiable, then $\Pi \cup \{\delta(t)\}$ for a term t, where t is substituable into $\delta(X)$, is also satisfiable. We can keep the choice of a particular t open as long as possible by introducing a fresh Skolem function, which depends only on the free variables in the formula.

$$\frac{\exists X \delta(X)}{\delta(f(X_1, \ldots, X_n))} \, (\exists)$$

where $X_1, \ldots, X_n$ are all free variables occurring in $\exists X \delta(X)$ and f is a fresh function symbol not occurring in the tableau.
$\delta(f(X_1, \ldots, X_n))$ means that all free occurrences of X in $\delta(X)$ are replaced by $f(X_1, \ldots, X_n)$.

Other cases

There is no need to produce normal forms, however, if we start with a formula in a NNF, then the previous rules are enough.

Or we can implicitly use the following standard equalities

$$\begin{aligned}
\neg(\varphi \vee \psi) &\equiv \neg\varphi \wedge \neg\psi && (\wedge)\\
\neg(\varphi \to \psi) &\equiv \varphi \wedge \neg\psi && (\wedge)\\
\varphi \to \psi &\equiv \neg\varphi \vee \psi && (\vee)\\
\neg(\varphi \wedge \psi) &\equiv \neg\varphi \vee \neg\psi && (\vee)\\
\neg(\exists X\varphi) &\equiv \forall X(\neg\varphi) && (\forall)\\
\neg(\forall X\varphi) &\equiv \exists X(\neg\varphi) && (\exists)\\
\neg\neg\varphi &\equiv \varphi
\end{aligned}$$

as in NNF and add a new rule for every such case.

Closure rule (C-rule)

The rules we have seen just expand formulae, the closure rule is destructive in the sense that it modifies the whole tableau.

If there are two literals p and $\neg q$ on a branch in the tableau such that $\sigma = mgu(p, q)$, then we can apply σ on the whole tableau. Hence we close the branch with p and $\neg q$, because we have two complementary literals $p\sigma$ and $\neg q\sigma$ on it.

Note that there is no need to require two complementary literals, we can close a branch with any two complementary formulae ψ and $\neg\chi$ such that $\sigma = mgu(\psi, \chi)$, because $\psi\sigma$ and $\neg\chi\sigma$ are together unsatisfiable.

Example I.

We want to prove $\forall X p(X) \wedge \forall X q(X) \rightarrow \forall X (p(X) \wedge q(X))$.

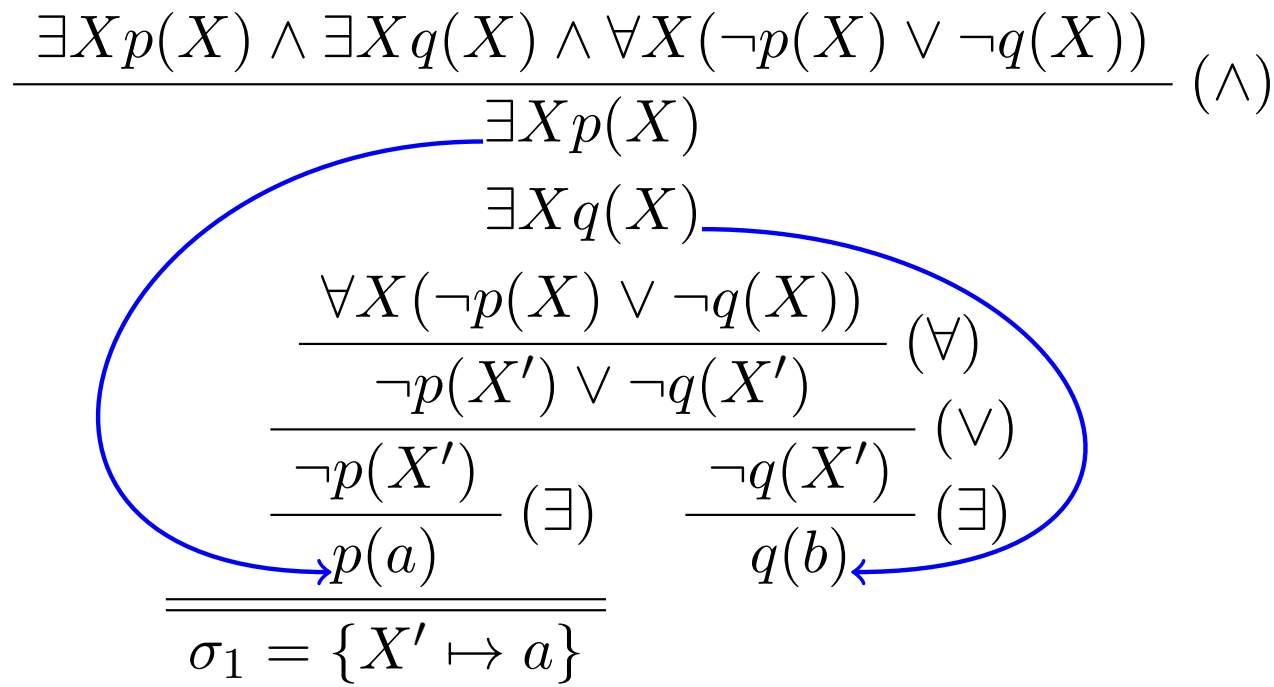
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall X p(X) \wedge \forall X q(X) \wedge \exists X (\neg p(X) \vee \neg q(X))}{\forall X p(X)}\,(\wedge)}{\forall X q(X)}}{\exists X (\neg p(X) \vee \neg q(X))}\,(\exists)}{\neg p(a) \vee \neg q(a)}\,(\vee)}{\dfrac{\neg p(a)}{p(X_1)}\,(\forall)} \qquad \dfrac{\dfrac{\neg q(a)}{q(X_2)}\,(\forall)}{}$$

$$\sigma_1 = \{X_1 \mapsto a\} \qquad \sigma_2 = \{X_2 \mapsto a\}$$

Note that $\sigma = \{X_1 \mapsto a, X_2 \mapsto a\}$, which is the composition of σ_1 and σ_2, is applied to the whole tableau.

Example II.

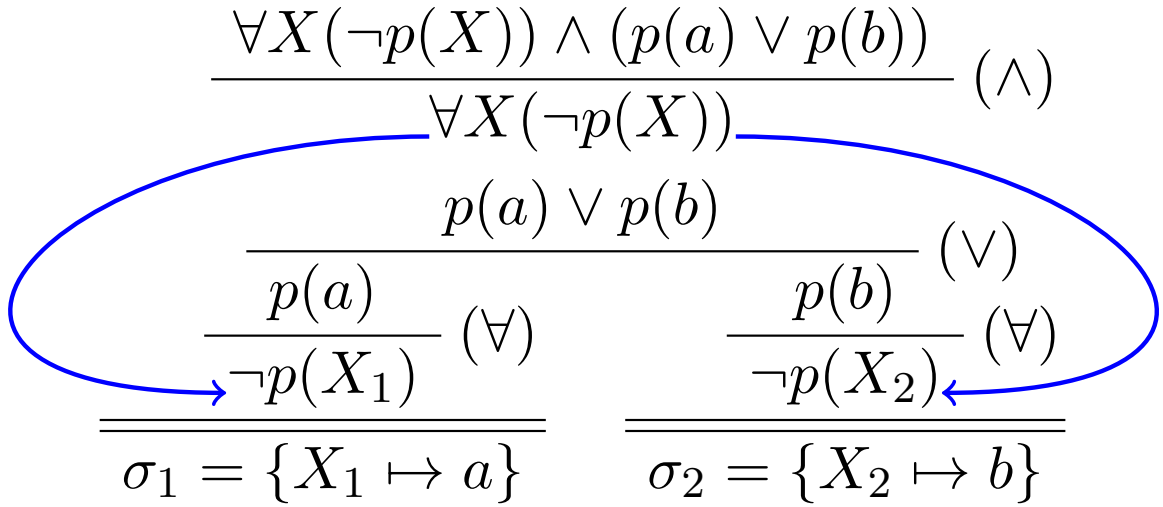
We want to prove $\exists X p(X) \wedge \exists X q(X) \rightarrow \exists X (p(X) \wedge q(X))$.



However, we cannot close the second branch by $\sigma_2 = \{X' \mapsto b\}$, because X' is already a after σ_1 is applied to the whole tableau! It also shows that we cannot select a in the second branch as a fresh constant on the branch, but it has to be a fresh constant in the whole tableau.

Example III.

We want to prove $\forall X(\neg p(X)) \to (\neg p(a) \wedge \neg p(b))$.



Note that we have to use $\forall X(\neg p(X))$ twice.

Proof search

Instead of describing a systematic proof procedure for a single free-variable tableaux, we enumerate all possible tableaux and a closed tableau, if it exists, is then among them.

We can use a breadth-first search or a depth-first search with backtracking and iterative deepening. The later variant is quite common in Prolog implementations.

There are many simple implementations of tableaux in Prolog, e.g., leanT^{A}P or leanCoP.

Equality and other extensions

It is possible to extend tableaux similarly to resolution. A common extension is an equality handling, see D'Agostino et al. 1999.

leanTAP

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

Tableaux systems

Note that there are many variants of tableaux; sometimes called semantic tableaux.

They are popular, for example, in non-classical logics, because

- no need for special normal forms like CNF,
  - can be complicated, or
  - even impossible to obtain
- given a semantic meaning of a connective we can usually produce a rule (or rules) in a straightforward way.

Generally, they are relatively easy to produce, in most cases, and still suitable for automated theorem proving.

Moreover, they are similar to other proof systems like natural deduction and sequent calculi.

Section 2

Finite model finding

How do we show that a formula is not provable?

We have seen several methods that can be used to prove a formula φ from a set of formulae Γ and hence $\Gamma \models \varphi$. However, can we use them to show that $\Gamma \not\models \varphi$? Sometimes we can, but it is quite rare, e.g., if we obtain a saturated set.

Note that $\Gamma \not\models \varphi$ is not equivalent to $\Gamma \models \neg\varphi$! For example, $\not\models p(a)$ and $\not\models \neg p(a)$.

A general method is to provide a counterexample. A model of Γ where φ is false, for simplicity assume that φ is a closed formula.

How do we find a counterexample?

We have to check all possible models.

Finite models

For a finite language and a given size of domain, it is possible to check all possible models exhaustively (up to trivial isomorphisms).

Infinite models

Clearly, there are many sets of formulae with only infinite models, for example,

$$\forall X \neg (X < X),$$
$$\forall X \forall Y \forall Z (X < Y \wedge Y < Z \rightarrow X < Z),$$
$$\forall X \exists Y (X < Y).$$

However, the problem how to generate useful infinite models is widely open. Moreover, for many problems finite counterexamples are sufficient.

MACE-style approach

We attempt to generate a finite counterexample iteratively. We try to produce a model of size 1, 2, 3, ...

The main idea is to produce a grounding of the problem assuming a given cardinality of our model and encode such a grounding as a SAT problem. Using a clever encoding we can significantly reduce the search space; no need to go through all possible models of the given size.

We present some basic techniques used in a model finder called Paradox, see Claessen and Sörensson 2003.

Our example

There is a counterexample for the problem that from

$$\begin{aligned} e \cdot X &= X, \\ X^{-1} \cdot X &= e, \\ X \cdot (Y \cdot Z) &= (X \cdot Y) \cdot Z \end{aligned}$$

follows

$$X \cdot (X \cdot X) = X.$$

Or equivalently.

Our example

There is a model for

$$e \cdot X = X, \tag{1}$$
$$X^{-1} \cdot X = e, \tag{2}$$
$$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z, \tag{3}$$
$$\neg(a \cdot (a \cdot a) = a). \tag{4}$$

We have $\mathcal{M} = (D, i)$, where $D = \{1, 2, 3\}$, $i(e) = 1$, $i(a) = 2$, and

$i(^{-1})$	
1	1
2	3
3	2

$i(\cdot)$	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

Propositional encoding

We are looking for a model $\mathcal{M} = (D, i)$ of a given cardinality, say n, that satisfies a set of clauses Γ. Assume without loss of generality that $D = \{1, \ldots, n\}$. Hence it only remains to generate a function i.

We want to describe i using propositional variables (atoms). For every k-ary

- predicate symbol p in Γ, there is a prop. variable for every

$$p(d_1, \ldots, d_k)$$

  where $d_1, \ldots, d_k \in D$.
- function symbol f in Γ, there is a prop. variable for every

$$f(d_1, \ldots, d_k) = d$$

  where $d_1, \ldots, d_k, d \in D$.

This is all we need to describe a model.

Our example

Note that our example is a bit confusing—we have only one predicate symbol ($=$), which is very special, because it has the fixed meaning. Still we have atoms for all

$$1 = 1, \quad 1 = 2, \quad 1 = 3, \quad 2 = 1, \quad \ldots, \quad 3 = 2, \quad 3 = 3$$

We also have propositional variables for all

$$e = 1, \quad e = 2, \quad e = 3$$
$$a = 1, \quad a = 2, \quad a = 3$$
$$1^{-1} = 1, \quad 1^{-1} = 2, \quad 1^{-1} = 3, \quad 2^{-1} = 1, \quad \ldots, \quad 3^{-1} = 3$$
$$1 \cdot 1 = 1, \quad 1 \cdot 1 = 2, \quad 1 \cdot 1 = 3, \quad 1 \cdot 2 = 1, \quad \ldots, \quad 3 \cdot 3 = 3$$

Flattening

However, it is impossible to express complex terms like $X \cdot (Y \cdot Z)$ directly in our language. We can only express so called shallow literals:

- $p(X_1, \ldots, X_k)$, or $\neg p(X_1, \ldots, X_k)$,
- $f(X_1, \ldots, X_l) = Y$, or $f(X_1, \ldots, X_l) \neq Y$,
- $X = Y$.

Note that $s \neq t$ is a shortcut for $\neg s = t$.

We do not want $X \neq Y$, because we can transform a clause

$$\varphi(X, Y) \vee X \neq Y$$

into

$$\varphi(X, X).$$

Note that a clause $\{X \neq Y\}$ is unsatisfiable.

Flattening complex terms

If we have a clause

$$\varphi(t),$$

it is equivalent to

$$\forall X(X = t \to \varphi(X)),$$

which is

$$X \neq t \vee \varphi(X)$$

where X is fresh in $\varphi(t)$. $\varphi(X)$ is produced from $\varphi(t)$ by replacing all (free) occurrences of t by X.

We can repeat this process as long as necessary.

Example

$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z \rightsquigarrow (X \cdot Y) \cdot Z \neq W \vee X \cdot (Y \cdot Z) = W \rightsquigarrow X \cdot Y \neq V \vee V \cdot Z \neq W \vee Y \cdot Z \neq U \vee X \cdot U = W.$

Instantiating

For every flattened clause we create three sets of propositional clauses

1. instances — we generate all possible groundings, where we can immediately simplify all groundings containing $d_1 = d_2$ or $d_1 \neq d_2$, for $d_1, d_2 \in D$, based on whether it is true (discard the clause), or not (discard the literal)
2. function definitions — for each k-ary function f and $d, d' \in D$ such that $d \neq d'$, we add

$$\{f(d_1, \ldots, d_k) \neq d, f(d_1, \ldots, d_k) \neq d'\}$$

   for every $d_1, \ldots, d_k \in D$.
3. totality definitions — for each k-ary function f, we add

$$\{f(d_1, \ldots, d_k) = 1, f(d_1, \ldots, d_k) = 2, \ldots, f(d_1, \ldots, d_k) = n\}$$

   for every $d_1, \ldots, d_k \in D$.

Reducing the number of distinct variables

The number of instances is exponential in the number of distinct variables in a flattened clause.

Term definitions

It is possible to decrease the number of newly introduced variables during flattening by using definitions based on constants. From $a \cdot (a \cdot a)$ we can obtain $a \cdot b$, where b is a fresh constant, and define $b = a \cdot a$. It is also possible to introduce definitions for non-ground terms and use definitions across clauses.

Clause splitting

If a clause can be split into parts, where each part contains less distinct variables than the whole clause, then we can decrease the number of distinct variables by introducing a new predicate.

Example

From $\{p(X, Y), q(Y, Z)\}$, we can produce $\{p(X, Y), r(Y)\}, \{\neg r(Y), q(Y, Z)\}$, where r is a fresh predicate.

Isomorphic models

We have $\mathcal{M} = (D, i)$, where $D = \{1, 2, 3\}$, $i(e) = 1$, $i(z) = 2$, and

$i(^{-1})$	
1	1
2	3
3	2

$i(\cdot)$	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

Note that any permutation on elements of D produces an isomorphic model. It makes no sense to look for all of them.

Static symmetry reduction

It is possible to avoid many isomorphic models using the following symmetry reduction technique. If we start to build a model by interpreting a constant c_1, then we can safely assign $i(c_1) = 1$, because no element of D has an assigned meaning. Hence we have

$$\{c_1 = 1\} \text{ instead of } \{c_1 = 1, c_1 = 2, \ldots, c_1 = n\}$$

Then we can assume that $i(c_2) \in \{1, 2\}$ and $i(c_3) \in \{1, 2, 3\}$, because it has to be interpreted by an element with a meaning, or the first fresh element (if available). However, if $i(c_2) = 1$, then $i(c_3) \in \{1, 2\}$. Or more generally

$$\{c_i \neq k, c_1 = k - 1, c_2 = k - 1, \ldots c_{i-1} = k - 1\}$$

This can be used also for functions, however, we have to take into account the meaning assigned to elements of D by constants.

Example

Hence $i(e) = 1$ and $i(a) = 2$ in our example, although $i(a) = 3$ works as well.

Other techniques

It is possible to use other techniques like

- pre-processing in SAT—variable and clause elimination, which is incompatible with an incremental search
- finding bounds for $|D|$
  - look for cardinality axioms
  - EPR—no function symbols and hence $|D|$ is bounded by the number of constants occurring in the problem
- use sorts
  - some problems are expressed in a many-sorted language,
  - other problems can be reformulated in a many-sorted language, if we have parts that can be defined independently

Bibliography I


Claessen, Koen and Niklas Sörensson (2003). "New Techniques that Improve MACE-style Finite Model Finding". In: *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*. Ed. by Peter Baumgartner and Chris Fermüller.


D'Agostino, Marcello et al., eds. (1999). *Handbook of Tableau Methods*. Springer Netherlands. DOI: 10.1007/978-94-017-1754-0.


Robinson, John Alan and Andrei Voronkov, eds. (2001). *Handbook of Automated Reasoning*. Vol. 1. Elsevier Science.