Propositional and Predicate Logic - III

Petr Gregor

KTIML MFF UK

WS 2024/2025

Algebra of propositions

Let T be a consistent theory over $\mathbb{P}$. On the quotient set $\mathrm{PF}_{\mathbb{P}}/\sim_T$ we define operations $\neg$, $\wedge$, $\vee$, $\perp$, $\top$ (correctly) by use of representatives, e.g.

$$[\varphi]_{\sim_T} \wedge [\psi]_{\sim_T} = [\varphi \wedge \psi]_{\sim_T}$$

Then $AV^{\mathbb{P}}(T) = \langle \mathrm{PF}_{\mathbb{P}}/\sim_T, \neg, \wedge, \vee, \perp, \top \rangle$ is *algebra of propositions* for T.

Since $\varphi \sim_T \psi \;\Leftrightarrow\; M(T,\varphi) = M(T,\psi)$, it follows that $h([\varphi]_{\sim_T}) = M(T,\varphi)$ is a (well-defined) injective function $h\colon \mathrm{PF}_{\mathbb{P}}/\sim_T \to \mathcal{P}(M(T))$ and

$$\begin{aligned} h(\neg[\varphi]_{\sim_T}) &= M(T) \setminus M(T,\varphi) \\ h([\varphi]_{\sim_T} \wedge [\psi]_{\sim_T}) &= M(T,\varphi) \cap M(T,\psi) \\ h([\varphi]_{\sim_T} \vee [\psi]_{\sim_T}) &= M(T,\varphi) \cup M(T,\psi) \\ h([\perp]_{\sim_T}) = \emptyset, &\quad h([\top]_{\sim_T}) = M(T) \end{aligned}$$

Moreover, h is *surjective* if $M(T)$ is *finite*.

Corollary If T is a consistent theory over a finite $\mathbb{P}$, then $AV^{\mathbb{P}}(T)$ is a Boolean algebra *isomorphic* via h to the (finite) algebra of sets $\underline{\mathcal{P}}(M(T))$.

Analysis of theories over finite languages

Let T be a consistent theory over $\mathbb{P}$ where $|\mathbb{P}| = n \in \mathbb{N}^+$ and $m = |M^{\mathbb{P}}(T)|$.
Then the number of (mutually) nonequivalent

- propositions (or theories) over $\mathbb{P}$ is 2^{2^n},
- propositions over $\mathbb{P}$ that are valid (contradictory) in T is $2^{2^n - m}$,
- propositions over $\mathbb{P}$ that are independent in T is $2^{2^n} - 2.2^{2^n - m}$,
- simple extensions of T is 2^m, out of which 1 is inconsistent,
- complete simple extensions of T is m.

And the number of (mutually) T-nonequivalent

- propositions over $\mathbb{P}$ is 2^m,
- propositions over $\mathbb{P}$ that are valid (contradictory) (in T) is 1,
- propositions over $\mathbb{P}$ that are independent (in T) is $2^m - 2$.

Proof By the bijection of $\mathrm{PF}_{\mathbb{P}}/\sim$ resp. $\mathrm{PF}_{\mathbb{P}}/\sim_T$ with $\mathcal{P}(M(\mathbb{P}))$ resp. $\mathcal{P}(M^{\mathbb{P}}(T))$ it suffices to determine the number of appropriate subsets of models. $\square$

SAT problem and solvers

- Problem SAT: Is φ in CNF satisfiable?
- *Example Is it possible to perfectly cover the chessboard without two diagonally removed corners using the domino tiles?*

  We can easily form a propositional formula that is satisfiable, if and only if the answer is yes. Then we can test its satisfiability by a SAT solver.
- Best SAT solvers: `www.satcompetition.org`.
- SAT solver in the demo: `Glucose`, CNF format: `DIMACS`.
- *Can all the mathematics be translated into logical formulas?*
  AI, theorem proving, Peano: *Formulario* (1895-1908), Mizar, LEAN
- How can we solve it more *elegantly*? What is our approach based on?

2-SAT

- A proposition in CNF is in *k-CNF* if every its clause has at most k literals.
- *k-SAT* is the problem of satisfiability of a given proposition in k-CNF.

Although for $k = 3$ it is an NP-complete problem, we show that 2-SAT can be solved in *linear* time (with respect to the length of φ).

We neglect implementation details (computational model, representation in memory) and we use the following fact, see [ADS I].

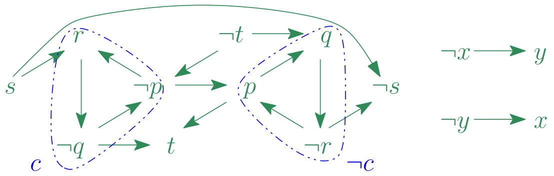
Proposition *A partition of a directed graph (V, E) to strongly connected components can be found in time $\mathcal{O}(|V| + |E|)$.*

- A directed graph G is *strongly connected* if for every two vertices u and v there are directed paths in G both from u to v and from v to u.
- A strongly connected *component* of a graph G is a maximal strongly connected subgraph of G.

Implication graphs

An *implication graph* of a proposition φ in 2-CNF is a directed graph G_φ s.t.

- vertices are all the propositional letters in φ and their negations,
- a clause $l_1 \vee l_2$ in φ is represented by a pair of edges $\overline{l_1} \to l_2$, $\overline{l_2} \to l_1$,
- a clause l_1 in φ is represented by an edge $\overline{l_1} \to l_1$.



$$p \wedge (\neg p \vee q) \wedge (\neg q \vee \neg r) \wedge (p \vee r) \wedge (r \vee \neg s) \wedge (\neg p \vee t) \wedge (q \vee t) \wedge \neg s \wedge (x \vee y)$$

Proposition *φ is satisfiable if and only if no strongly connected component of G_φ contains a pair of complementary literals.*

Proof Every satisfying assignment assigns the same value to all the literals in a same component. Thus the implication from left to right holds (necessity).

Satisfying assignment

For the implication from right to left (sufficiency), let G^*_φ be the graph obtained from G_φ by contracting strongly connected components to single vertices.

Observation *G^*_φ is acyclic, and therefore has a topological ordering $<$.*

- A directed graph is *acyclic* if it is has no directed *cycles*.
- A linear ordering $<$ of vertices of a directed graph is *topological* if $p < q$ for every edge from p to q.

Now for every unassigned component in an increasing order by $<$, assign 0 to all its literals and 1 to all literals in the complementary component.

It remains to show that such assignment v satisfies φ. If not, then G^*_φ contains edges $p \to q$ and $\overline{q} \to \overline{p}$ with $v(p) = 1$ and $v(q) = 0$. But this contradicts the order of assigning values to components since $p < q$ and $\overline{q} < \overline{p}$. □

Corollary *2-SAT can be solved in a linear time.*

Horn-SAT

- A *unit clause* is a clause containing a single literal,
- a *Horn clause* is a clause containing at most one positive literal,

$$\neg p_1 \vee \cdots \vee \neg p_n \vee q \quad \sim \quad (p_1 \wedge \cdots \wedge p_n) \rightarrow q$$

- a *Horn formula* is a conjunction of Horn clauses,
- *Horn-SAT* is the problem of satisfiability of a given Horn formula.

Algorithm

(1) *if φ contains a pair of unit clauses l and $\overline{l}$, then it is not satisfiable,*

(2) *if φ contains a unit clause l, then assign 1 to l, remove all clauses containing l, remove $\overline{l}$ from all clauses, and repeat from the start,*

(3) *if φ does not contain a unit clause, then it is satisfied by assigning 0 to all remaining propositional variables.*

Step (2) is called *unit propagation*.

Unit propagation

$$\begin{aligned}
&(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge (\neg r \vee \neg s) \wedge (\neg t \vee s) \wedge s && v(s) = 1\\
&(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge \neg r && v(\neg r) = 1\\
&(\neg p \vee q) \wedge (\neg p \vee \neg q) && v(p) = v(q) = v(t) = 0
\end{aligned}$$

Observation *Let φ^l be the proposition obtained from φ by unit propagation. Then φ^l is satisfiable if and only if φ is satisfiable.*

Corollary *The algorithm is correct (it solves Horn-SAT).*

Proof The correctness in Step (1) is obvious, in Step (2) it follows from the observation, in Step (3) it follows from the *Horn form* since every remaining clause contains at least one negative literal.

Note A direct implementation requires quadratic time, but with an appropriate representation in memory, one can achieve linear time (w.r.t. the length of φ).

DPLL algorithm

A literal l is *pure* in a CNF formula φ if l occurs in φ and $\overline{l}$ does not occur in φ.

Algorithm DPLL(φ)

(1) *while φ contains a unit clause l, assign 1 to l, remove all clauses containing l, remove $\overline{l}$ from all clauses, and repeat, (unit propagation)*

(2) *while φ contains a pure literal l, assign 1 to l, remove all clauses containing l and repeat, (pure literal elimination)*

(3) *if φ contains an empty clause, then it is not satisfiable,*

(4) *if φ does not contain any clause, then it is satisfiable,*

(5) *choose an unassigned propositional letter p and run DPLL($\varphi \wedge p$) and DPLL($\varphi \wedge \neg p$). (branching)*

Note The algoritm runs in exponentional time in the worst case. Its correctness is easy to verify.

Formal proof systems

We formalize precisely the notion of proof as a syntactical procedure.

In *(standard)* formal proof systems,

- a proof is a finite object, it can be built from axioms of a given theory,
- $T \vdash \varphi$ denotes that φ is *provable* from a theory T,
- if a formula has a proof, it can be found *“algorithmically”*,
  (If T is *“given algorithmically”*.)

We usually require that a formal proof system is

- *sound*, i.e. every formula provable from a theory T is also valid in T,
- *complete*, i.e. every formula valid in T is also provable from T.

Examples of formal proof systems (calculi): tableaux methods, resolution, *Hilbert systems, Gentzen systems, natural deduction systems*.

Tableau method - introduction

We assume that the language is fixed and countable, i.e. the set $\mathbb{P}$ of propositional letters is countable. Then every theory over $\mathbb{P}$ is countable.

Main features of the tableau method *(informally)*

- a tableau for a formula φ is a binary labeled tree representing systematic search for *counterexample* to φ, i.e. a model of theory is which φ is false,
- a formula is proved if every branch in tableau 'fails', i.e counterexample was not found. In this case the (systematic) tableau will be finite,
- if a counterexample exists, there will be a branch in a (finished) tableau that provides us with this counterexample, but this branch can be infinite.

Introductory examples

$$\begin{tikzcd}
 & F(((p \to q) \to p) \to p) \arrow[d, dash] & \\
 & T((p \to q) \to p) \arrow[d, dash] & \\
 & Fp \arrow[d, dash] & \\
 & T((p \to q) \to p) \arrow[dl, dash] \arrow[dr, dash] & \\
F(p \to q) \arrow[d, dash] & & Tp \arrow[d, dash] \\
F(p \to q) \arrow[d, dash] & & \otimes \\
Tp \arrow[d, dash] & & \\
Fq \arrow[d, dash] & & \\
\otimes & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & F((\neg q \vee p) \to p) \arrow[d, dash] & \\
 & T(\neg q \vee p) \arrow[d, dash] & \\
 & Fp \arrow[d, dash] & \\
 & T(\neg q \vee p) \arrow[dl, dash] \arrow[dr, dash] & \\
T(\neg q) \arrow[d, dash] & & Tp \arrow[d, dash] \\
T(\neg q) \arrow[d, dash] & & \otimes \\
Fq & &
\end{tikzcd}$$

Explanation to examples

Nodes in tableaux are labeled by *entries*. An entry is a formula with a *sign* T / F representing an assumption that the formula is true / false in some model. If this assumption is correct, then it is correct also for all the entries in some branch below that came from this entry.

In both examples we have finished (systematic) tableaux from no axioms.

- On the left, there is a *tableau proof* for $((p \to q) \to p) \to p$. All branches *"failed"*, denoted by $\otimes$, as each contains a pair $T\varphi$, $F\varphi$ for some φ *(counterexample was not found)*. Thus the formula is provable, written by
$$\vdash ((p \to q) \to p) \to p$$
- On the right, there is a (finished) tableau for $(\neg q \vee p) \to p$. The left branch did not *"fail"* and is finished (all its entries were considered) *(it provides us with a counterexample $v(p) = v(q) = 0$)*.

Atomic tableaux

An *atomic tableau* is one of the following trees (labeled by entries), where p is any propositional letter and φ, ψ are any propositions.

Tp	Fp	$\begin{tikzcd} T(\varphi \land \psi) \arrow[d, dash] \\ T\varphi \arrow[d, dash] \\ T\psi \end{tikzcd}$	$\begin{tikzcd} & F(\varphi \land \psi) \arrow[dl, dash] \arrow[dr, dash] & \\ F\varphi & & F\psi \end{tikzcd}$	$\begin{tikzcd} & T(\varphi \lor \psi) \arrow[dl, dash] \arrow[dr, dash] & \\ T\varphi & & T\psi \end{tikzcd}$	$\begin{tikzcd} F(\varphi \lor \psi) \arrow[d, dash] \\ F\varphi \arrow[d, dash] \\ F\psi \end{tikzcd}$
$\begin{tikzcd} T(\neg\varphi) \arrow[d, dash] \\ F\varphi \end{tikzcd}$	$\begin{tikzcd} F(\neg\varphi) \arrow[d, dash] \\ T\varphi \end{tikzcd}$	$\begin{tikzcd} & T(\varphi \to \psi) \arrow[dl, dash] \arrow[dr, dash] & \\ F\varphi & & T\psi \end{tikzcd}$	$\begin{tikzcd} F(\varphi \to \psi) \arrow[d, dash] \\ T\varphi \arrow[d, dash] \\ F\psi \end{tikzcd}$	$\begin{tikzcd} & T(\varphi \leftrightarrow \psi) \arrow[dl, dash] \arrow[dr, dash] & \\ T\varphi \arrow[d, dash] & & F\varphi \arrow[d, dash] \\ T\psi & & F\psi \end{tikzcd}$	$\begin{tikzcd} & F(\varphi \leftrightarrow \psi) \arrow[dl, dash] \arrow[dr, dash] & \\ T\varphi \arrow[d, dash] & & F\varphi \arrow[d, dash] \\ F\psi & & T\psi \end{tikzcd}$

All tableaux will be formally defined with atomic tableaux and rules how to expand them.

Tableaux

A *finite tableau* is a binary tree labeled with entries described (inductively) by

(i) every atomic tableau is a finite tableau,

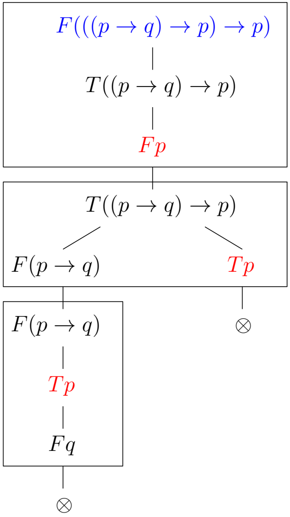
(ii) if P is an entry on a branch V in a finite tableau τ and τ' is obtained from τ by adjoining the atomic tableaux for P at the end of branch V, then τ' is also a finite tableau,

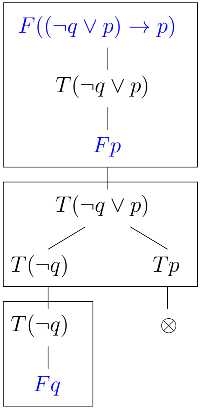
(iii) every finite tableau is formed by a finite number of steps (i), (ii).

A *tableau* is a sequence $\tau_0, \tau_1, \ldots, \tau_n, \ldots$ (finite or infinite) of finite tableaux such that τ_{n+1} is formed from τ_n by an application of (ii), formally $\tau = \cup \tau_n$.

Remark *It is not specified how to choose the entry P and the branch V for expansion. This will be specified in systematic tableaux.*

Construction of tableaux





Convention

$$\begin{tikzcd}
 & F(((p \to q) \to p) \to p) \arrow[d, dash] & \\
 & T((p \to q) \to p) \arrow[d, dash] & \\
 & Fp \arrow[dl, dash] \arrow[dr, dash] & \\
F(p \to q) \arrow[d, dash] & & Tp \arrow[d, dash] \\
Tp \arrow[d, dash] & & \otimes \\
Fq \arrow[d, dash] & & \\
\otimes & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & F((\neg q \vee p) \to p) \arrow[d, dash] & \\
 & T(\neg q \vee p) \arrow[d, dash] & \\
 & Fp \arrow[dl, dash] \arrow[dr, dash] & \\
T(\neg q) \arrow[d, dash] & & Tp \arrow[d, dash] \\
Fq & & \otimes
\end{tikzcd}$$

We will not write the entry that is expanded again on the branch.

Remark *They will actually be needed later in predicate tableau method.*

Tableau proofs

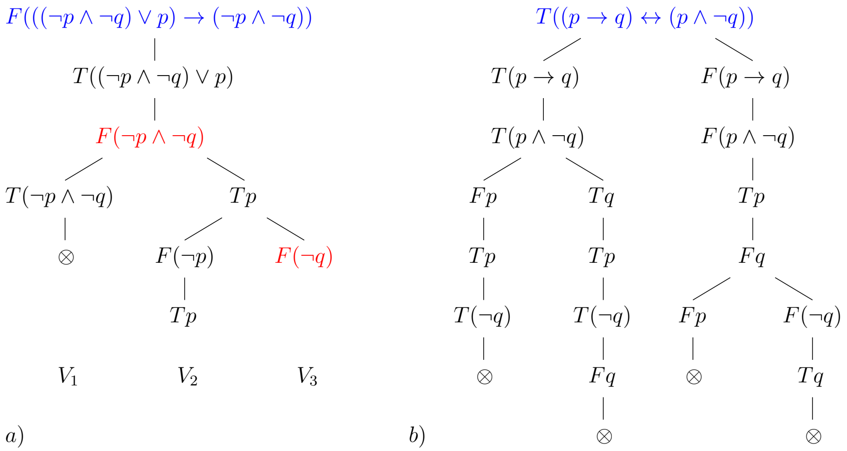
Let P be an entry on a branch V in a tableau τ. We say that

- the entry P is *reduced* on V if it occurs on V as a root of an atomic tableau, i.e. it was already expanded on V during the construction of τ,
- the branch V is *contradictory* if it contains entries $T\varphi$ and $F\varphi$ for some proposition φ, otherwise V is *noncontradictory*. The branch V is *finished* if it is contradictory or every entry on V is already reduced on V,
- the tableau τ is *finished* if every branch in τ is finished, and τ is *contradictory* if every branch in τ is contradictory.

A *tableau proof* (*proof by tableau*) of φ is a contradictory tableau with the root entry $F\varphi$. φ is *(tableau) provable*, denoted by $\vdash \varphi$, if it has a tableau proof.

Similarly, a *refutation* of φ by *tableau* is a contradictory tableau with the root entry $T\varphi$. φ is *(tableau) refutable* if it has a refutation by tableau, i.e. $\vdash \neg\varphi$.

Examples



a) $F(\neg p \wedge \neg q)$ not reduced on V_1, V_1 contradictory, V_2 finished, V_3 unfinished,

b) a (tableau) refutation of φ: $(p \rightarrow q) \leftrightarrow (p \wedge \neg q)$, i.e. $\vdash \neg\varphi$.