Decision Procedures for Verification

Part 1. Propositional Logic (1)

31.10.2016

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Part 1: Propositional Logic

Literature (also for first-order logic)

Schöning: Logik für Informatiker, Spektrum

Fitting: First-Order Logic and Automated Theorem Proving, Springer

Part 1: Propositional Logic

Propositional logic

- logic of truth values
- decidable (but NP-complete)
- can be used to describe functions over a finite domain
- important for hardware applications (e.g., model checking)

1.1 Syntax

- propositional variables
- logical symbols
  $\Rightarrow$ Boolean combinations

Propositional Variables

Let Π be a set of propositional variables.

We use letters P, Q, R, S, to denote propositional variables.

Propositional Formulas

F_Π is the set of propositional formulas over Π defined as follows:

$$
\begin{array}{llll}
F, G, H & ::= & \bot & \text{(falsum)} \\
& \mid & \top & \text{(verum)} \\
& \mid & P, \quad P \in \Pi & \text{(atomic formula)} \\
& \mid & \neg F & \text{(negation)} \\
& \mid & (F \wedge G) & \text{(conjunction)} \\
& \mid & (F \vee G) & \text{(disjunction)} \\
& \mid & (F \rightarrow G) & \text{(implication)} \\
& \mid & (F \leftrightarrow G) & \text{(equivalence)}
\end{array}
$$

Notational Conventions

- We omit brackets according to the following rules:
  - $\neg \quad >_p \quad \wedge \quad >_p \quad \vee \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$ (binding precedences
  - $\vee$ and $\wedge$ are associative and commutative

1.2 Semantics

In classical logic (dating back to Aristoteles) there are “only” two truth values “true” and “false” which we shall denote, respectively, by 1 and 0.

There are multi-valued logics having more than two truth values.

Valuations

A propositional variable has no intrinsic meaning. The meaning of a propositional variable has to be defined by a valuation.

A Π-valuation is a map

$$\mathcal{A} : \Pi \to \{0, 1\}.$$

where $\{0, 1\}$ is the set of truth values.

Truth Value of a Formula in $\mathcal{A}$

Given a Π-valuation $\mathcal{A}$, the function $\mathcal{A}^*$: Σ-formulas $\to \{0, 1\}$ is defined inductively over the structure of F as follows:

$$\mathcal{A}^*(\bot) = 0$$

$$\mathcal{A}^*(\top) = 1$$

$$\mathcal{A}^*(P) = \mathcal{A}(P)$$

$$\mathcal{A}^*(\neg F) = 1 - \mathcal{A}^*(F)$$

$$\mathcal{A}^*(F \rho G) = \mathrm{B}_\rho(\mathcal{A}^*(F), \mathcal{A}^*(G))$$

with B_ρ the Boolean function associated with ρ

For simplicity, we write $\mathcal{A}$ instead of $\mathcal{A}^*$.

Truth Value of a Formula in $\mathcal{A}$

Example: Let's evaluate the formula

$$(P \to Q) \wedge (P \wedge Q \to R) \to (P \to R)$$

w.r.t. the valuation $\mathcal{A}$ with

$$\mathcal{A}(P) = 1, \mathcal{A}(Q) = 0, \mathcal{A}(R) = 1$$

(On the blackboard)

1.3 Models, Validity, and Satisfiability

F is valid in $\mathcal{A}$ ($\mathcal{A}$ is a model of F; F holds under $\mathcal{A}$):

$$\mathcal{A} \models F \;:\Leftrightarrow\; \mathcal{A}(F) = 1$$

F is valid (or is a tautology):

$$\models F \;:\Leftrightarrow\; \mathcal{A} \models F \text{ for all } \Pi\text{-valuations } \mathcal{A}$$

F is called satisfiable iff there exists an $\mathcal{A}$ such that $\mathcal{A} \models F$.
Otherwise F is called unsatisfiable (or contradictory).

A set N of formulae is satisfiable iff there exists an $\mathcal{A}$ such that $\mathcal{A} \models F$ for all $F \in N$.
Otherwise N is called unsatisfiable (or contradictory).

Example

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

A	B	C	$(A \vee C)$	$\neg C$	$(B \vee \neg C)$	$(A \vee C) \wedge (B \vee \neg C)$
0	0	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	1	1	1	0	1	1
1	0	0	1	1	1	1
1	0	1	1	0	0	0
1	1	0	1	1	1	1
1	1	1	1	0	1	1

Let $\mathcal{A} : \{A, B, C\} \to \{0, 1\}$ with $\mathcal{A}(A) = 0, \mathcal{A}(B) = 1, \mathcal{A}(C) = 1$.

$\mathcal{A} \models (A \vee C), \quad \mathcal{A} \models (B \vee \neg C)$
$\mathcal{A} \models (A \vee C) \wedge (B \vee \neg C)$
$\mathcal{A} \models \{(A \vee C), (B \vee \neg C)\}$

1.3 Models, Validity, and Satisfiability

Examples:

$F \rightarrow F$ and $F \vee \neg F$ are valid for all formulae F.

Obviously, every valid formula is also satisfiable

$F \wedge \neg F$ is unsatisfiable

The formula P is satisfiable, but not valid

Example

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

A	B	C	$(A \vee C)$	$\neg C$	$(B \vee \neg C)$	$(A \vee C) \wedge (B \vee \neg C)$
0	0	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	1	1	1	0	1	1
1	0	0	1	1	1	1
1	0	1	1	0	0	0
1	1	0	1	1	1	1
1	1	1	1	0	1	1

F is not valid:

$\mathcal{A}_1(F) = 0$ für $\mathcal{A}_1 : \{A, B, C\} \to \{0, 1\}$ mit $\mathcal{A}(A) = \mathcal{A}(B) = \mathcal{A}(C) = 0$.

F is satisfiable:

$\mathcal{A}_2(F) = 1$ für $\mathcal{A} : \{A, B, C\} \to \{0, 1\}$ mit $\mathcal{A}(A) = 0, \mathcal{A}(B) = 1, \mathcal{A}(C) = 1$.

Entailment and Equivalence

F entails (implies) G (or G is a consequence of F), written $F \models G$, if for all Π-valuations $\mathcal{A}$, whenever $\mathcal{A} \models F$ then $\mathcal{A} \models G$.

F and G are called equivalent if for all Π-valuations $\mathcal{A}$ we have $\mathcal{A} \models F \Leftrightarrow \mathcal{A} \models G$.

Example

$F = (A \vee C) \wedge (B \vee \neg C) \qquad G = (A \vee B)$

Check if $F \models G$

A	B	C	$(A \vee C)$	$(B \vee \neg C)$	$(A \vee C) \wedge (B \vee \neg C)$	$(A \vee B)$
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Example

$F = (A \vee C) \wedge (B \vee \neg C)$ $\qquad$ $G = (A \vee B)$

Check if $F \models G$

A	B	C	$(A \vee C)$	$(B \vee \neg C)$	$(A \vee C) \wedge (B \vee \neg C)$	$(A \vee B)$
0	0	0	0	1	0	0
0	0	1	1	0	0	0
0	1	0	0	1	0	1
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	1	0	0	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Example

$F = (A \vee C) \wedge (B \vee \neg C) \qquad G = (A \vee B)$

Check if $F \models G$ Yes, $F \models G$

A	B	C	$(A \vee C)$	$(B \vee \neg C)$	$(A \vee C) \wedge (B \vee \neg C)$	$(A \vee B)$
0	0	1	1	0	0	0
0	0	0	0	1	0	0
0	1	1	1	1	1	1
0	1	0	0	1	0	1
1	0	1	1	0	0	1
1	0	0	1	1	1	1
1	1	1	1	1	1	1
1	1	0	1	1	1	1

Example

$F = (A \vee C) \wedge (B \vee \neg C) \qquad G = (A \vee B)$

Check if $F \models G$ Yes, $F \models G$

... But it is not true that $G \models F$ (Notation: $G \not\models F$)

A	B	C	$(A \vee C)$	$(B \vee \neg C)$	$(A \vee C) \wedge (B \vee \neg C)$	$(A \vee B)$
0	0	1	1	0	0	0
0	0	0	0	1	0	0
0	1	1	1	1	1	1
0	1	0	0	1	0	1
1	0	1	1	0	0	1
1	0	0	1	1	1	1
1	1	1	1	1	1	1
1	1	0	1	1	1	1

Entailment and Equivalence

F entails (implies) G (or G is a consequence of F), written $F \models G$, if for all Π-valuations $\mathcal{A}$, whenever $\mathcal{A} \models F$ then $\mathcal{A} \models G$.

F and G are called equivalent if for all Π-valuations $\mathcal{A}$ we have $\mathcal{A} \models F \Leftrightarrow \mathcal{A} \models G$.

Proposition 1.1:
F entails G iff $(F \rightarrow G)$ is valid

Proposition 1.2:
F and G are equivalent iff $(F \leftrightarrow G)$ is valid.

Entailment and Equivalence

Extension to sets of formulas N in the "natural way", e.g., $N \models F$ if for all Π-valuations $\mathcal{A}$: if $\mathcal{A} \models G$ for all $G \in N$, then $\mathcal{A} \models F$.

Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 1.3:

$$F \text{ valid} \Leftrightarrow \neg F \text{ unsatisfiable}$$

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

Q: In a similar way, entailment $N \models F$ can be reduced to unsatisfiability. How?

Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 1.4:

$$N \models F \Leftrightarrow N \cup \{\neg F\} \text{ unsatisfiable}$$

Hence in order to design a theorem prover (validity/entailment checker) it is sufficient to design a checker for unsatisfiability.

Checking Unsatisfiability

Every formula F contains only finitely many propositional variables. Obviously, $\mathcal{A}(F)$ depends only on the values of those finitely many variables in F under $\mathcal{A}$.

If F contains n distinct propositional variables, then it is sufficient to check 2^n valuations to see whether F is satisfiable or not.
$\Rightarrow$ truth table.

So the satisfiability problem is clearly decidable
(but, by Cook's Theorem, NP-complete).

Nevertheless, in practice, there are (much) better methods than truth tables to check the satisfiability of a formula. (later more)

Checking Unsatisfiability

The satisfiability problem is clearly decidable
(but, by Cook's Theorem, NP-complete).

For sets of propositional formulae of a certain type, satisfiability can be checked in polynomial time:

Examples: 2SAT, Horn-SAT (will be discussed in the exercises)

Dichotomy theorem. Schaefer [Schaefer, STOC 1978] identified six classes of sets S of Boolean formulae for which $SAT(S)$ is in PTIME. He proved that all other types of sets of formulae yield an NP-complete problem.

Substitution Theorem

Proposition 1.5:

Let F and G be equivalent formulas, let H be a formula in which F occurs as a subformula.

Then H is equivalent to H' where H' is obtained from H by replacing the occurrence of the subformula F by G.
(Notation: $H = H[F]$, $H' = H[G]$.)

Proof: By induction over the formula structure of H.

Some Important Equivalences

Proposition 1.6:

The following equivalences are valid for all formulas F, G, H:

$$(F \wedge F) \leftrightarrow F$$

$$(F \vee F) \leftrightarrow F \qquad \text{(Idempotency)}$$

$$(F \wedge G) \leftrightarrow (G \wedge F)$$

$$(F \vee G) \leftrightarrow (G \vee F) \qquad \text{(Commutativity)}$$

$$(F \wedge (G \wedge H)) \leftrightarrow ((F \wedge G) \wedge H)$$

$$(F \vee (G \vee H)) \leftrightarrow ((F \vee G) \vee H) \qquad \text{(Associativity)}$$

$$(F \wedge (G \vee H)) \leftrightarrow ((F \wedge G) \vee (F \wedge H))$$

$$(F \vee (G \wedge H)) \leftrightarrow ((F \vee G) \wedge (F \vee H)) \qquad \text{(Distributivity)}$$

Some Important Equivalences

Proposition 1.7:

The following equivalences are valid for all formulas F, G, H:

$(F \wedge (F \vee G)) \leftrightarrow F$	
$(F \vee (F \wedge G)) \leftrightarrow F$	(Absorption)
$(\neg\neg F) \leftrightarrow F$	(Double Negation)
$\neg(F \wedge G) \leftrightarrow (\neg F \vee \neg G)$	
$\neg(F \vee G) \leftrightarrow (\neg F \wedge \neg G)$	(De Morgan's Laws)
$(F \wedge G) \leftrightarrow F$, if G is a tautology	
$(F \vee G) \leftrightarrow \top$, if G is a tautology	(Tautology Laws)
$(F \wedge G) \leftrightarrow \bot$, if G is unsatisfiable	
$(F \vee G) \leftrightarrow F$, if G is unsatisfiable	(Tautology Laws)

1.4 Normal Forms

We define conjunctions of formulas as follows:

$\bigwedge_{i=1}^{0} F_i = \top.$

$\bigwedge_{i=1}^{1} F_i = F_1.$

$\bigwedge_{i=1}^{n+1} F_i = \bigwedge_{i=1}^{n} F_i \wedge F_{n+1}.$

and analogously disjunctions:

$\bigvee_{i=1}^{0} F_i = \bot.$

$\bigvee_{i=1}^{1} F_i = F_1.$

$\bigvee_{i=1}^{n+1} F_i = \bigvee_{i=1}^{n} F_i \vee F_{n+1}.$

Literals and Clauses

A literal is either a propositional variable P or a negated propositional variable $\neg P$.

A clause is a (possibly empty) disjunction of literals.

Literals and Clauses

A literal is either a propositional variable P or a negated propositional variable $\neg P$.

A clause is a (possibly empty) disjunction of literals.

Example of clauses:

$\bot$	the empty clause
P	positive unit clause
$\neg P$	negative unit clause
$P \vee Q \vee R$	positive clause
$P \vee \neg Q \vee \neg R$	clause
$P \vee P \vee \neg Q \vee \neg R \vee R$	allow repetitions/complementary literals

CNF and DNF

A formula is in conjunctive normal form (CNF, clause normal form), if it is a conjunction of disjunctions of literals (or in other words, a conjunction of clauses).

A formula is in disjunctive normal form (DNF), if it is a disjunction of conjunctions of literals.

Warning: definitions in the literature differ:

are complementary literals permitted?
are duplicated literals permitted?
are empty disjunctions/conjunctions permitted?

CNF and DNF

Checking the validity of CNF formulas or the unsatisfiability of DNF formulas is easy:

A formula in CNF is valid, if and only if each of its disjunctions contains a pair of complementary literals P and $\neg P$.

Conversely, a formula in DNF is unsatisfiable, if and only if each of its conjunctions contains a pair of complementary literals P and $\neg P$.

On the other hand, checking the unsatisfiability of CNF formulas or the validity of DNF formulas is known to be coNP-complete.

Conversion to CNF/DNF

Proposition 1.8:

For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).

Proof:

We consider the case of CNF.

Apply the following rules as long as possible (modulo associativity and commutativity of $\wedge$ and $\vee$):

Step 1: Eliminate equivalences:

$$(F \leftrightarrow G) \;\Rightarrow_K\; (F \rightarrow G) \wedge (G \rightarrow F)$$

Conversion to CNF/DNF

Step 2: Eliminate implications:

$$(F \to G) \;\Rightarrow_K\; (\neg F \vee G)$$

Step 3: Push negations downward:

$$\neg(F \vee G) \;\Rightarrow_K\; (\neg F \wedge \neg G)$$
$$\neg(F \wedge G) \;\Rightarrow_K\; (\neg F \vee \neg G)$$

Step 4: Eliminate multiple negations:

$$\neg\neg F \;\Rightarrow_K\; F$$

The formula obtained from a formula F after applying steps 1-4 is called the negation normal form (NNF) of F

Conversion to CNF/DNF

Step 5: Push disjunctions downward:

$$(F \wedge G) \vee H \;\Rightarrow_K\; (F \vee H) \wedge (G \vee H)$$

Step 6: Eliminate $\top$ and $\bot$:

$$\begin{aligned}
(F \wedge \top) &\;\Rightarrow_K\; F \\
(F \wedge \bot) &\;\Rightarrow_K\; \bot \\
(F \vee \top) &\;\Rightarrow_K\; \top \\
(F \vee \bot) &\;\Rightarrow_K\; F \\
\neg\bot &\;\Rightarrow_K\; \top \\
\neg\top &\;\Rightarrow_K\; \bot
\end{aligned}$$

Conversion to CNF/DNF

Proving termination is easy for most of the steps; only step 3 and step 5 are a bit more complicated.

The resulting formula is equivalent to the original one and in CNF.

The conversion of a formula to DNF works in the same way, except that disjunctions have to be pushed downward in step 5.

Complexity

Conversion to CNF (or DNF) may produce a formula whose size is exponential in the size of the original one.

Satisfiability-preserving Transformations

The goal

"find a formula G in CNF such that $\models F \leftrightarrow G$"

is unpractical.

But if we relax the requirement to

"find a formula G in CNF such that $F \models \bot$ iff $G \models \bot$"

we can get an efficient transformation.

Satisfiability-preserving Transformations

Idea:
A formula $F[F']$ is satisfiable iff $F[P] \wedge (P \leftrightarrow F')$ is satisfiable (where P new propositional variable that works as abbreviation for F').

We can use this rule recursively for all subformulas in the original formula (this introduces a linear number of new propositional variables).

Conversion of the resulting formula to CNF increases the size only by an additional factor (each formula $P \leftrightarrow F'$ gives rise to at most one application of the distributivity law).

Optimized Transformations

A further improvement is possible by taking the polarity of the subformula F into account.

Assume that F contains neither $\rightarrow$ nor $\leftrightarrow$. A subformula F' of F has positive polarity in F, if it occurs below an even number of negation signs; it has negative polarity in F, if it occurs below an odd number of negation signs.

Optimized Transformations

Proposition 1.9:
Let $F[F']$ be a formula containing neither $\rightarrow$ nor $\leftrightarrow$; let P be a propositional variable not occurring in $F[F']$.

If F' has positive polarity in F, then $F[F']$ is satisfiable if and only if $F[P] \wedge (P \rightarrow F')$ is satisfiable.

If F' has negative polarity in F, then $F[F']$ is satisfiable if and only if $F[P] \wedge (F' \rightarrow P)$ is satisfiable.

Proof:
Exercise.

This satisfiability-preserving transformation to clause form is also called structure-preserving transformation to clause form.

Optimized Transformations

Example: Let $F = (Q_1 \wedge Q_2) \vee (R_1 \wedge R_2)$.

The following are equivalent:

- $F \models \bot$
- $P_F \;\wedge\; (P_F \leftrightarrow (P_{Q_1 \wedge Q_2} \vee P_{R_1 \wedge R_2}) \;\wedge\; (P_{Q_1 \wedge Q_2} \leftrightarrow (Q_1 \wedge Q_2))$
  $\wedge\; (P_{R_1 \wedge R_2} \leftrightarrow (R_1 \wedge R_2)) \models \bot$
- $P_F \;\wedge\; (P_F \rightarrow (P_{Q_1 \wedge Q_2} \vee P_{R_1 \wedge R_2}) \;\wedge\; (P_{Q_1 \wedge Q_2} \rightarrow (Q_1 \wedge Q_2))$
  $\wedge\; (P_{R_1 \wedge R_2} \rightarrow (R_1 \wedge R_2)) \models \bot$
- $P_F \;\wedge\; (\neg P_F \vee P_{Q_1 \wedge Q_2} \vee P_{R_1 \wedge R_2}) \;\wedge\; (\neg P_{Q_1 \wedge Q_2} \vee Q_1) \wedge (\neg P_{Q_1 \wedge Q_2} \vee Q_2)$
  $\wedge\; (\neg P_{R_1 \wedge R_2} \vee R_1) \wedge (\neg P_{R_1 \wedge R_2} \vee R_2)) \models$

Decision Procedures for Satisfiability

- Simple Decision Procedures

  truth table method

- The Resolution Procedure

- The Davis-Putnam-Logemann-Loveland Algorithm

1.5 Inference Systems and Proofs

Inference systems Γ (proof calculi) are sets of tuples

$$(F_1, \ldots, F_n, F_{n+1}), \;\; n \geq 0,$$

called inferences or inference rules, and written

$$\frac{\overbrace{F_1 \;\; \ldots \;\; F_n}^{\text{premises}}}{\underbrace{F_{n+1}}_{\text{conclusion}}} \; .$$

Clausal inference system: premises and conclusions are clauses. One also considers inference systems over other data structures.

Proofs

A proof in Γ of a formula F from a a set of formulas N (called assumptions) is a sequence $F_1, \ldots, F_k$ of formulas where

(i) $F_k = F$,

(ii) for all $1 \leq i \leq k$: $F_i \in N$, or else there exists an inference $(F_{i_1}, \ldots, F_{i_{n_i}}, F_i)$ in Γ, such that $0 \leq i_j < i$, for $1 \leq j \leq n_i$.

Soundness and Completeness

Provability $\vdash_\Gamma$ of F from N in Γ:

$N \vdash_\Gamma F$ $\;:\Leftrightarrow\;$ there exists a proof Γ of F from N.

Γ is called sound $:\Leftrightarrow$

$$\frac{F_1 \;\; \ldots \;\; F_n}{F} \in \Gamma \quad \Rightarrow \quad F_1, \ldots, F_n \models F$$

Γ is called complete $:\Leftrightarrow$

$$N \models F \quad \Rightarrow \quad N \vdash_\Gamma F$$

Γ is called refutationally complete $:\Leftrightarrow$

$$N \models \bot \quad \Rightarrow \quad N \vdash_\Gamma \bot$$