

Non-classical logics

Lecture 1: Classical logic

29.10.2014

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Winter Semester 2014/2015

Part 1: Propositional Logic

Literature (also for first-order logic)

Schöning: Logik für Informatiker, Spektrum

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

1.1 Syntax

- propositional variables
- logical symbols
 - ⇒ 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:

F, G, H	$::=$	\perp	(falsum)
		\top	(verum)
		$P, P \in \Pi$	(atomic formula)
		$\neg F$	(negation)
		$(F \wedge G)$	(conjunction)
		$(F \vee G)$	(disjunction)
		$(F \rightarrow G)$	(implication)
		$(F \leftrightarrow G)$	(equivalence)

Notational Conventions

- We omit brackets according to the following rules:

– $\neg > \wedge > \vee > \rightarrow > \leftrightarrow$ (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

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

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 $\mathcal{A}^* : \Sigma\text{-formulas} \rightarrow \{0, 1\}$ is defined inductively over the structure of F as follows:

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

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

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

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

$$\mathcal{A}^*(F \rho G) = B_\rho(\mathcal{A}^*(F), \mathcal{A}^*(G))$$

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

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

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**).

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**

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 $\mathcal{A} \models F$.

Definition A set N of formulae is satisfiable if there exists a Π -valuation \mathcal{A} which makes true all formulae in N .

If there is no Π -valuation \mathcal{A} which makes true all formulae in N we say that N is unsatisfiable

Remark: N unsatisfiable iff $N \models \perp$.

Validity vs. Unsatisfiability

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

Proposition 1.3:

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

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

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)

Some Important Equivalences

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

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

$$(F \vee F) \leftrightarrow F$$

(Idempotency)

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

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

(Commutativity)

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

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

(Associativity)

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

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

(Distributivity)

Some Important Equivalences

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)

$$(F \wedge G) \leftrightarrow F, \text{ if } G \text{ is a tautology}$$

$$(F \vee G) \leftrightarrow \top, \text{ if } G \text{ is a tautology}$$

(Tautology Laws)

$$(F \wedge G) \leftrightarrow \perp, \text{ if } G \text{ is unsatisfiable}$$

$$(F \vee G) \leftrightarrow F, \text{ if } G \text{ is unsatisfiable}$$

(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**:

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

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

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

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:

\perp

P

$\neg P$

$P \vee Q \vee R$

$P \vee \neg Q \vee \neg R$

$P \vee P \vee \neg Q \vee \neg R \vee R$

the empty clause

positive unit clause

negative unit clause

positive clause

clause

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?

Conversion to CNF/DNF

Proposition 1.4:

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:

$$(F \rightarrow G) \Rightarrow_K (\neg F \vee G)$$

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 \perp :

$$(F \wedge \top) \Rightarrow_K F$$

$$(F \wedge \perp) \Rightarrow_K \perp$$

$$(F \vee \top) \Rightarrow_K \top$$

$$(F \vee \perp) \Rightarrow_K F$$

$$\neg \perp \Rightarrow_K \top$$

$$\neg \top \Rightarrow_K \perp$$

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 $F \models \perp$ iff $G \models \perp$ ”

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 1.5:

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:

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

Overview

Propositional Logic

- Syntax
- Semantics

Validity, Satisfiability, Entailment, Equivalence

Conjunctive and Disjunctive Normal Forms

Transformation to CNF

- Using distributivity
- Structure-preserving transformation + optimization