Classical Propositional Logic

Software Formal Verification

Maria João Frade

Departmento de Informática
Universidade do Minho

2008/2009

Introduction

- The language of propositional logic is based on propositions, or declarative sentences which one can, in principle, argue as being "true" or "false".

  "The capital of Portugal is Braga."

  "D. Afonso Herriques was the first king of Portugal."

- Propositions are the atomic formulas of the language. More complex sentences are constructed using logical connectives.
- In classical propositional logic (PL) each sentence is either true or false.
- In fact, the content of the propositions is not relevant to PL. PL is not the study of truth, but of the relationship between the truth of one statement and that of another.

Syntax

- *Propositional variables*: $P, Q, R, \ldots \in \mathcal{V}_{\text{Prop}}$ (a countably infinite set)
- *Logical connectives*: $\bot, \top, \neg, \wedge, \vee, \rightarrow$

The set $\mathcal{S}$ of *formulas* of propositional logic is given by the abstract syntax

$$\mathcal{S} \ni A, B \ ::= \ P \mid \bot \mid \top \mid \neg A \mid A \wedge B \mid A \vee B \mid A \rightarrow B$$

We let $A, B, C, F, G, H, \ldots$ range over $\mathcal{S}$.

$\neg$ binds more tightly than $\vee$ and $\wedge$, and the latter two bind more tightly than $\rightarrow$. Implication $\rightarrow$ is right-associative: expressions of the form $P \rightarrow Q \rightarrow R$ denote $P \rightarrow (Q \rightarrow R)$.

Semantics

The semantics of a logic provides its meaning. What exactly is meaning? In propositional logic, meaning is given by the truth values true and false, where true $\neq$ false. We will represent true by 1 and false by 0.

An *assignment* is a function $\mathcal{A} : \mathcal{V}_{\mathsf{Prop}} \to \{0, 1\}$, that assigns to every propositional variable truth value.
An assignment $\mathcal{A}$ naturally extends to all formulas, $\mathcal{A} : \mathcal{S} \to \{0, 1\}$. The truth value of a formula is computed using *truth tables*:

F	A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \to B$	$\perp$	$\top$
$\mathcal{A}_1(F)$	0	1	1	0	1	1	0	1
$\mathcal{A}_2(F)$	0	0	1	0	0	1	0	1
$\mathcal{A}_3(F)$	1	1	0	1	1	1	0	1
$\mathcal{A}_4(F)$	1	0	0	0	1	0	0	1

Semantics

Let $\mathcal{A}$ be an assignment and let F be a formula.
If $\mathcal{A}(F) = 1$, then we say F *holds* under assignment $\mathcal{A}$, or $\mathcal{A}$ *models* F.
We write $\mathcal{A} \models F$ iff $\mathcal{A}(F) = 1$, and $\mathcal{A} \not\models F$ iff $\mathcal{A}(F) = 0$.

An alternative (inductive) definition of $\mathcal{A} \models F$ is

$$
\begin{array}{lll}
\mathcal{A} \models \top & & \\
\mathcal{A} \not\models \bot & & \\
\mathcal{A} \models P & \text{iff} & \mathcal{A}(P) = 1 \\
\mathcal{A} \models \neg A & \text{iff} & \mathcal{A} \not\models A \\
\mathcal{A} \models A \wedge B & \text{iff} & \mathcal{A} \models A \text{ and } \mathcal{A} \models B \\
\mathcal{A} \models A \vee B & \text{iff} & \mathcal{A} \models A \text{ or } \mathcal{A} \models B \\
\mathcal{A} \models A \rightarrow B & \text{iff} & \mathcal{A} \not\models A \text{ or } \mathcal{A} \models B
\end{array}
$$

Validity, satisfiability, and contradiction

A formula F is

valid	iff	it holds under every assignment. We write $\models F$. A valid formula is called a *tautology*.
satisfiable	iff	it holds under some assignment.
unsatisfiable	iff	it holds under no assignment. An unsatisfiable formula is called a *contradiction*.

Proposition

F is valid iff $\neg F$ is a contradiction

- $P \vee \neg P$ is a tautology.
- $A \wedge \neg A$ is a contradiction.
- $A \rightarrow B$ is satisfiable.
- $(A \wedge (A \rightarrow B)) \rightarrow B$ is valid.

Consequence and equivalence

- $F \models G$ iff for every assignment $\mathcal{A}$, if $\mathcal{A} \models F$ then $\mathcal{A} \models G$. We say G is a *consequence* of F.

- $F \equiv G$ iff $F \models G$ and $F \models G$. We say F and G are *equivalent*.

- Let $\Gamma = \{F_1, F_2, F_3, \dots\}$ be a set of formulas.
  $\mathcal{A} \models \Gamma$ iff $\mathcal{A} \models F_i$ for each formula F_i in Γ. We say $\mathcal{A}$ *models* Γ.
  $\Gamma \models G$ iff $\mathcal{A} \models \Gamma$ implies $\mathcal{A} \models G$ for every assignment $\mathcal{A}$. We say G is a *consequence* of Γ.

Proposition

- $F \models G$ iff $\models F \rightarrow G$
- $\Gamma \models G$ and Γ finite iff $\models \bigwedge \Gamma \rightarrow G$

Some basic equivalences

$$\begin{array}{lcl}
A \vee A & \equiv & A \\
A \wedge A & \equiv & A \\
\\
A \vee B & \equiv & B \vee A \\
A \wedge B & \equiv & B \wedge A \\
\\
A \wedge (A \vee B) & \equiv & A \\
\\
A \wedge (B \vee C) & \equiv & (A \wedge B) \vee (A \wedge C) \\
A \vee (B \wedge C) & \equiv & (A \vee B) \wedge (A \vee C) \\
\\
\neg(A \vee B) & \equiv & \neg A \wedge \neg B \\
\neg(A \wedge B) & \equiv & \neg A \vee \neg B
\end{array}$$

$$\begin{array}{lcl}
A \wedge \neg A & \equiv & \bot \\
A \vee \neg A & \equiv & \top \\
\\
A \wedge \top & \equiv & A \\
A \vee \top & \equiv & \top \\
\\
A \wedge \bot & \equiv & \bot \\
A \vee \bot & \equiv & A \\
\\
\neg\neg A & \equiv & A \\
\\
A \rightarrow B & \equiv & \neg A \vee B
\end{array}$$

Consistency

Let $\Gamma = \{F_1, F_2, F_3, \dots\}$ be a set of formulas.

- Γ is *consistent* or *satisfiable* iff there is an assignment that models Γ.
- We say that Γ is *inconsistent* iff it is not consistent and denote this by $\Gamma \models \bot$.

Proposition

- $\{F, \neg F\} \models \bot$
- If $\Gamma \models \bot$ and $\Gamma \subseteq \Delta$, then $\Delta \models \bot$.
- $\Gamma \models F \;$ iff $\; \Gamma, \neg F \models \bot$

Theories

A set of formulas $\mathcal{T}$ is *closed* under logical consequence iff for all formulas F, if $\mathcal{T} \models F$ then $F \in \mathcal{T}$.

$\mathcal{T}$ is a *theory* iff it is closed under logical consequence. The elements of $\mathcal{T}$ are called *theorems*.

Let Γ be a set of formulas.

$\mathcal{T}(\Gamma) = \{F \mid \Gamma \models F\}$ is called the *theory* of Γ.
The formulas of Γ are called *axioms* and the theory $\mathcal{T}(\Gamma)$ is *axiomatizable*.

Substitution

- Formula G is a *subformula* of formula F if it occurs syntactically within F.
- Formula G is a *strict subformula* of F if G is a subformula of F and $G \neq F$

Substitution theorem

Suppose $F \equiv G$. Let H be a formula that contains F as a subformula. Let H' be the formula obtained by replacing some occurrence of F in H with G. Then $H \equiv H'$.

Adquate sets of connectives for PL

There is some redundancy among the logical connectives.

Some smaller adquate sets of conectives for PL:

$\{\wedge, \neg\}$ $\quad \bot \equiv P \wedge \neg P, \quad \top \equiv \neg(P \wedge \neg P),$
$A \vee B \equiv \neg(\neg A \wedge \neg B), \quad A \to B \equiv \neg(A \wedge \neg B)$

$\{\vee, \neg\}$ $\quad \top \equiv A \vee \neg A, \quad \bot \equiv \neg(A \vee \neg A),$
$A \wedge B \equiv \neg(\neg A \vee \neg B), \quad A \to B \equiv A \vee B$

$\{\to, \neg\}$ $\quad \top \equiv A \to A, \quad \bot \equiv \neg(A \to A),$
$A \vee B \equiv \neg A \to B, \quad A \wedge B \equiv \neg(A \to \neg B)$

$\{\to, \bot\}$ $\quad \neg A \equiv A \to \bot, \quad \top \equiv A \to A,$
$A \vee B \equiv (A \to \bot) \to B), \quad A \wedge B \equiv (A \to B \to \bot) \to \bot$

Decidability

A *decision problem* is any problem that, given certain input, asks a question to be answered with a "yes" or a "no".

A *solution* to a decision problem is a program that takes problem instances as input and always terminates, producing a correct "yes" or "no" output. A decision problem is *decidable* if it has a solution.

Given formulas F and G as input, we may ask:

Decision problems

Validity problem:	"Is F valid ?"
Satisfiability problem:	"Is F satisfiable ?"
Consequence problem:	"Is G a consequence of F ?"
Equivalence problem:	"Are F and G equivalent ?"

All these problems are **decidable!**

Decidability

Any algorithm that works for one of these problems also works for all of these problems!

F is satisfiable	iff	$\neg F$ is not valid
$F \models G$	iff	$\neg(F \to G)$ is not satisfiable
$F \equiv G$	iff	$F \models G$ and $G \models F$
F is valid	iff	$F \equiv \top$

Truth-table method

For the Satisfiability problem, we first compute a truth table for F and then check to see if its truth value is ever one.

This algorithm certainly works, but is very inefficient.
Its exponential-time! $\mathcal{O}(2^n)$

If F has n atomic formulas, then the truth table for F has 2^n rows.

Complexity

An algorithm is *polynomial-time* if there exists a polynomial $p(x)$ such that given input of size n, the algorithm halts in fewer than $p(n)$ steps. The class of all decision problems that can be resolved by some polynomial-time algorithm is denoted by **P** (or PTIME).

It is not known whether the Satisfiability problem (and the other three decision problems) is in **P**.

We do not know of a polynomial-time algorithm for satisfiability.

If it exists, then **P** = **NP** !

The Satisfiability problem for PL (PSAT) is **NP**-complete.

Complexity

- A *deterministic* algorithm is a step-by-step procedure. At any stage of the algorithm, the next step is completely determined.
- In contrast, a *nondeterministic* algorithm may have more than one possible "next step" at a given stage. That is, there may be more than one computation for a given input.

NP *(nondeterministic polynomial-time)* decision problems

Let PROB be an arbitrary decision problem. Given certain input, PROB produces an output of either "yes" or "no". Let Y be the set of all inputs for which PROB produces the output of "yes" and let N be the analogous set of inputs that produce output "no".

- If there exists a nondeterministic algorithm which, given input x, can produce the output "yes" in polynomial-time if and only if $x \in Y$, then PROB is in **NP**.
- If there exists a nondeterministic algorithm which, given input x, can produce the output "no" in polynomial-time if and only if $x \in N$, then PROB is in **coNP**.

Complexity

Essentially, a decision problem is in **NP** (**coNP**) if a "yes" ("no") answer can be obtained in polynomial-time by guessing.

Satisfiability problem is **NP**.

Given a formula F compute *an* assignment $\mathcal{A}$ for F.
If $\mathcal{A}(F) = 1$, then F is satisfiable.

Validity problem is **co-NP**.

NP-complete

A decision problem Π is **NP**-*complete* if it is in **NP** and for every problem Π_1 in **NP**, Π_1 is *polynomially reducible* to Π ($\Pi_1 \propto \Pi$).

Cook's theorem (1971)

PSAT is **NP**-complete.

Decision Procedures for Satisfiability

We have seen the truth-table method for deciding satisfiability.

However this method is very inefficient (exponential on the number of propositions).

The are more efficient algorithms (although not polynomial-time) for the decision problems presented above:

- Semantic Tableaux
- Resolution
- Davis-Putnam-Logemann-Loveland algorithm (DPLL)

Proof system

A logic, by definition, has rules for deducing the truth of one sentence from that of another. These rules yield a system of formal proof.

A *proof system* consists of a set of basic rules for derivations. These rules allow us to deduce formulas from sets of formulas.

- It may take several steps to derive a given formula G from a set of formulas Γ, where each step is an application of one of the basic rules. The list of these steps forms a formal proof of G from Γ.
- We write: $\Gamma \vdash G$ to abbreviate *"G can be derived from Γ"*.
- We want a proof system that is sound: if $\Gamma \vdash G$, then $\Gamma \models G$.

Natural deduction in sequent style

- The proof system we introduce is called *Natural Deduction*, because the rules embody patterns familiar in ordinary reasoning, or at least in mathematical proof.
- The statement $\Gamma \vdash G$ is called a *sequent*. Γ represent the set of hypothesis and current assumptions, and is called the *context* of the sequent.
- We will present Natural Deduction in sequent style.
  - It is potentate a clear representation of the discharging of assumptions.
  - It is closer to what one gets while developing a proof in a proof-assistant.

Natural deduction

- The set of basic rules provided is intended to aid the translation of thought (mathematical reasoning) into formal proof.
- This system is intended for human use, in the sense that
  - a person can guide the proof process;
  - the proof produced is highly legible, and easy to understand.

This contrast with decision procedures that just produce a "yes/no" answer, and may not give insight into the relationship between the assumption and the conclusion.

Other proof systems for PL: *Hilbert system*, *sequent calculus*.

Natural deduction

The veracity of the basic rules for the proof system is self-evident.

For example, if F and G can be derived from Γ, then $F \wedge G$ can also be derived from Γ .

This is the "$\wedge$-introduction" rule

$$\frac{\Gamma \vdash F \quad \Gamma \vdash G}{\Gamma \vdash F \wedge G} \wedge_I$$

There are two "$\wedge$-elimination" rules:

$$\frac{\Gamma \vdash F \wedge G}{\Gamma \vdash F} \wedge_{E1} \qquad\qquad \frac{\Gamma \vdash F \wedge G}{\Gamma \vdash G} \wedge_{E2}$$

Natural deduction rules for $\mathcal{S}$ (in sequent style)

$$\frac{}{\Gamma \vdash \top}\ \text{true} \qquad \frac{A \in \Gamma}{\Gamma \vdash A}\ \text{assumption} \qquad \frac{\Gamma \vdash A \quad \Gamma \subset \Gamma'}{\Gamma' \vdash A}\ \text{monotonicity}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge_I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge_{E1} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge_{E2}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee_{I1} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee_{I2} \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee_E$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to_I \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash A \to B}{\Gamma \vdash B}\ \to_E$$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \neg_I \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \bot}\ \neg_E$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot_E \qquad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}\ \neg\neg_E$$

Formal proof

Deduction is purely syntactical.

A *formal proof* is a finite sequence of statements of the form "$\Gamma \vdash F$" each of which follows from the previous statements by one of the basic rules. We say that G can be derived from Γ if there is a formal proof concluding with the statement $\Gamma \vdash G$.

Example: $\vdash A \wedge B \rightarrow A \vee B$

	Statements	Justification
1.	$A \wedge B \vdash A \wedge B$	assumption
2.	$A \wedge B \vdash A$	$\wedge_{E1}$ 1
3.	$A \wedge B \vdash A \vee B$	$\vee_I$ 2
4.	$\vdash A \wedge B \rightarrow A \vee B$	$\rightarrow_I$ 3

In a proof-assistant the proof is usually developed backwards.

In a proof-assistant

In a proof-assistant, the usual approach develop the proof by a method that is known as *goal directed proof*:

1. The user enters a statement that he wants to prove.
2. The system displays the formula as a formula to be proved, possibly giving a context of local facts that can be used for this proof.
3. The user enters a command (a basic rule or a *tactic*) to decompose the goal into simpler ones.
4. The system displays a list of formulas that still need to be proved.

When there are no more goals the proof is complete!

Formal proof

Note that rule $\neg\neg_E$ is ok because we are in a classical setting. Using this rule we can derive a proof of $A \vee \neg A$.

Example: $\vdash A \vee \neg A$

	Statements	Justification
1.	$\neg(A \vee \neg A), A \vdash A$	assumption
2.	$\neg(A \vee \neg A), A \vdash A \vee \neg A$	$\vee_{I1}$ 1
3.	$\neg(A \vee \neg A), A \vdash \neg(A \vee \neg A)$	assumption
4.	$\neg(A \vee \neg A), A \vdash \perp$	$\neg_E$ 2, 3
5.	$\neg(A \vee \neg A) \vdash \neg A$	$\neg_I$ 4
6.	$\neg(A \vee \neg A) \vdash A \vee \neg A$	$\vee_{I2}$ 5
7.	$\neg(A \vee \neg A) \vdash \perp$	$\neg_E$ 5, 3
8.	$\vdash \neg\neg(A \vee \neg A)$	$\neg_I$ 7
9.	$\vdash A \vee \neg A$	$\neg\neg_E$ 8

Formal proof

Note that from the basic deduction rules we can derive many other rules. We call these new rules *derivable* (or *admissible*).

Example of a derivable rule

	Statements	Justification
1.	$\Gamma \vdash A \wedge B$	premise
2.	$\Gamma \vdash A$	$\wedge_{E1}$ 1
3.	$\Gamma \vdash B$	$\wedge_{E2}$ 1
4.	$\Gamma \vdash B \wedge A$	$\wedge_{I}$ 3, 2

$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B \wedge A}$ is a derivable rule.

Formal proof

The following derivable rule is usually implemented in proof-assistants.

Example of a derivable rule

	Statements	Justification
1.	$\Gamma \vdash A \wedge B$	premise
2.	$\Gamma, A, B \vdash C$	premise
3.	$\Gamma \vdash A$	$\wedge_{E1}$ 1
4.	$\Gamma \vdash B$	$\wedge_{E2}$ 1
5.	$\Gamma, A \vdash B \to C$	$\to_I$ 2
6.	$\Gamma \vdash A \to B \to C$	$\to_I$ 5
7.	$\Gamma \vdash B \to C$	$\to_E$ 3, 6
8.	$\Gamma \vdash C$	$\to_I$ 4, 7

$$\frac{\Gamma \vdash A \wedge B \quad \Gamma, A, B \vdash C}{\Gamma \vdash C}$$ is a derivable rule.

Soundness, completeness and compacteness

Soundness

If $\Gamma \vdash F$, then $\Gamma \models F$,

Therefore, if $\vdash A$, then A is a tautology; and if $\vdash \neg A$, then A is a contradiction.

Completeness

If $\Gamma \models F$, then $\Gamma \vdash F$,

Compactness

A set of formulas is satisfiable if and only if every finite subset is satisfiable.

Hence,

If a formula F is a consequence of a (possible infinite) set of formulas Γ then it is a consequence of a finite subset of Γ.