Formalizing propositional logic

Joris Roos

University of Wisconsin-Madison

Sommerakademie Leysin 2018

August 14, 2018

Overview

1. Introduction
2. Syntax and semantics
3. Normal forms

Introduction

Consider a propositional formula

$$((A \vee D) \rightarrow (D \vee \neg B)) \wedge \neg(A \leftarrow (B \vee C \wedge D)$$

Introduction

Consider a propositional formula

$$((A \vee D) \to (D \vee \neg B)) \wedge \neg(A \leftarrow (B \vee C \wedge D))$$

- Is it satisfiable? ("SAT")
- Is it a tautology?

Introduction

Consider a propositional formula

$$((A \vee D) \to (D \vee \neg B)) \wedge \neg(A \leftarrow (B \vee C \wedge D))$$

- Is it satisfiable? ("SAT")
- Is it a tautology?

In principle, "trivially" decidable by truth tables.

Introduction

Consider a propositional formula

$$((A \vee D) \to (D \vee \neg B)) \wedge \neg(A \leftarrow (B \vee C \wedge D))$$

- Is it satisfiable? ("SAT")
- Is it a tautology?

In principle, "trivially" decidable by truth tables.

Goal: Formalize the problem and program a computer to solve it (provably correctly and "efficiently", if possible).

Why?

- Logic puzzles: knights and knaves and co.

Why?

- Logic puzzles: knights and knaves and co.
- Circuit design: circuits *are* propositional formulas!

Why?

- Logic puzzles: knights and knaves and co.
- Circuit design: circuits *are* propositional formulas!
- Surprisingly many problems can be rephrased as SAT: e.g. primality

Why?

- Logic puzzles: knights and knaves and co.
- Circuit design: circuits *are* propositional formulas!
- Surprisingly many problems can be rephrased as SAT: e.g. primality
- SAT is a computationally *hard* problem (NP-complete).

Why?

- Logic puzzles: knights and knaves and co.
- Circuit design: circuits *are* propositional formulas!
- Surprisingly many problems can be rephrased as SAT: e.g. primality
- SAT is a computationally *hard* problem (NP-complete).
- Classical first-order theorem provers rely on SAT algorithms

1 Introduction

2 Syntax and semantics

3 Normal forms

Syntax

A propositional formula is a *string* (i.e. a list of symbols from a certain alphabet).

Syntax

A propositional formula is a *string* (i.e. a list of symbols from a certain alphabet).

Admissible symbols are:

- Constants: $\top$ (true), $\perp$ (false)

Syntax

A propositional formula is a *string* (i.e. a list of symbols from a certain alphabet).

Admissible symbols are:

- Constants: $\top$ (true), $\bot$ (false)
- Atomic propositions/literals: $P, Q, R, \cdots$

Syntax

A propositional formula is a *string* (i.e. a list of symbols from a certain alphabet).

Admissible symbols are:

- Constants: $\top$ (true), $\bot$ (false)
- Atomic propositions/literals: $P, Q, R, \cdots$
- Logical operators: $\neg, \wedge, \vee$

Syntax

A propositional formula is a *string* (i.e. a list of symbols from a certain alphabet).

Admissible symbols are:

- Constants: $\top$ (true), $\bot$ (false)
- Atomic propositions/literals: $P, Q, R, \cdots$
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,

Syntax

A propositional formula is a *string* (i.e. a list of symbols from a certain alphabet).

Admissible symbols are:

- Constants: $\top$ (true), $\bot$ (false)
- Atomic propositions/literals: $P, Q, R, \cdots$
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,

Definition

The set of *propositional formulas* is the smallest set $\mathbf{P}$ of strings such that

1. $\top, \bot, P, Q, R, \cdots \in \mathbf{P}$.
2. If $X \in \mathbf{P}$, then $\neg X \in \mathbf{P}$.
3. If $X, Y \in \mathbf{P}$, then $\vee(X, Y) \in \mathbf{P}$ and $\wedge(X, Y) \in \mathbf{P}$.

Syntax

A propositional formula is a *string* (i.e. a list of symbols from a certain alphabet).

Admissible symbols are:

- Constants: $\top$ (true), $\bot$ (false)
- Atomic propositions/literals: $P, Q, R, \cdots$
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,

Definition

The set of *propositional formulas* is the smallest set $\mathbf{P}$ of strings such that

1. $\top, \bot, P, Q, R, \cdots \in \mathbf{P}$.
2. If $X \in \mathbf{P}$, then $\neg X \in \mathbf{P}$.
3. If $X, Y \in \mathbf{P}$, then $\vee(X, Y) \in \mathbf{P}$ and $\wedge(X, Y) \in \mathbf{P}$.

Syntactic sugar

Not part of the basic formal syntax. Just convenience.

Syntactic sugar

Not part of the basic formal syntax. Just convenience.

- Write $(X \vee Y)$ for $\vee(X, Y)$ and $(X \wedge Y)$ for $\wedge(X, Y)$.

Syntactic sugar

Not part of the basic formal syntax. Just convenience.

- Write $(X \vee Y)$ for $\vee(X, Y)$ and $(X \wedge Y)$ for $\wedge(X, Y)$.
- Brackets are ignored whenever possible according to usual operator precedence conventions ($\neg \gg \wedge \gg \vee$).

Syntactic sugar

Not part of the basic formal syntax. Just convenience.

- Write $(X \vee Y)$ for $\vee(X, Y)$ and $(X \wedge Y)$ for $\wedge(X, Y)$.
- Brackets are ignored whenever possible according to usual operator precedence conventions $(\neg \gg \wedge \gg \vee)$.
- $(X \to Y)$ is short for $(\neg X \vee Y)$.

Syntactic sugar

Not part of the basic formal syntax. Just convenience.

- Write $(X \vee Y)$ for $\vee(X, Y)$ and $(X \wedge Y)$ for $\wedge(X, Y)$.
- Brackets are ignored whenever possible according to usual operator precedence conventions $(\neg \gg \wedge \gg \vee)$.
- $(X \to Y)$ is short for $(\neg X \vee Y)$.
- may include other common logical operators, $\leftarrow, \leftrightarrow, \not\leftrightarrow, \cdots$

Example:

$$\vee(P, \neg \wedge (Q, P))$$

is in **P** (written sugarfree).

Example:

$$\vee(P, \neg \wedge (Q, P))$$

is in **P** (written sugarfree).
Same formula with sugar:

$$P \vee \neg(Q \wedge P)$$

We will always use sugar.

Example:

$$\vee(P, \neg \wedge (Q, P))$$

is in **P** (written sugarfree).
Same formula with sugar:

$$P \vee \neg(Q \wedge P)$$

We will always use sugar.
Non-example:

$$\wedge \vee (P, \neg Q())$$

is not in **P**.

Example:

$$\vee(P, \neg \wedge (Q, P))$$

is in **P** (written sugarfree).
Same formula with sugar:

$$P \vee \neg(Q \wedge P)$$

We will always use sugar.
Non-example:

$$\wedge \vee (P, \neg Q())$$

is not in **P**.

So far, formulas do not have any meaning!
A propositional formula is just a string with a certain structure.

Structural induction

Say we want to *prove* that every formula $F \in \mathbf{P}$ has a certain property Q.

Structural induction

Say we want to *prove* that every formula $F \in \mathbf{P}$ has a certain property Q. Then it suffices to show:

1. $\top, \bot, p, q, r, \ldots$ have property Q.
2. If F has property Q, then $\neg F$ has property Q.
3. If F and F' have property Q, then $F \vee F'$ and $F \wedge F'$ have property Q.

Structural induction

Say we want to *prove* that every formula $F \in \mathbf{P}$ has a certain property Q. Then it suffices to show:

1. $\top, \bot, p, q, r, \ldots$ have property Q.
2. If F has property Q, then $\neg F$ has property Q.
3. If F and F' have property Q, then $F \vee F'$ and $F \wedge F'$ have property Q.

This is a theorem! [Exercise: Prove it.]

Structural induction

Say we want to *prove* that every formula $F \in \mathbf{P}$ has a certain property Q. Then it suffices to show:

1. $\top, \bot, p, q, r, \ldots$ have property Q.
2. If F has property Q, then $\neg F$ has property Q.
3. If F and F' have property Q, then $F \vee F'$ and $F \wedge F'$ have property Q.

This is a theorem! [Exercise: Prove it.]
Example.
Exercise: Prove that no propositional formula consists entirely of the symbol $\neg$.

Semantics: Introducing meaning

Definition

A *valuation* v is a map that assigns each atomic proposition $P, Q, \cdots$ one of the truth values true or false:

$$v(P) \in \{\mathrm{true}, \mathrm{false}\}$$

Semantics: Introducing meaning

Definition

A *valuation* v is a map that assigns each atomic proposition $P, Q, \cdots$ one of the truth values true or false:

$$v(P) \in \{\mathrm{true}, \mathrm{false}\}$$

(Given n atomic propositions we have 2^n possible valuations.)

Semantics: Introducing meaning

Definition

A *valuation* v is a map that assigns each atomic proposition $P, Q, \cdots$ one of the truth values true or false:

$$v(P) \in \{\mathrm{true}, \mathrm{false}\}$$

(Given n atomic propositions we have 2^n possible valuations.)

Semantics: Introducing meaning

Let v be a valuation. We extend v to all propositional formulas:

Semantics: Introducing meaning

Let v be a valuation. We extend v to all propositional formulas:

- $v(\top) := \mathrm{true}, v(\bot) := \mathrm{false},$

Semantics: Introducing meaning

Let v be a valuation. We extend v to all propositional formulas:

- $v(\top) := \text{true}, v(\bot) := \text{false},$
- $v(\neg F)$ is true iff $v(F)$ is $\text{false}.$

Semantics: Introducing meaning

Let v be a valuation. We extend v to all propositional formulas:

- $v(\top) := \text{true}, v(\bot) := \text{false},$
- $v(\neg F)$ is true iff $v(F)$ is $\text{false}.$
- $v(F \vee F')$ is true iff at least one of $v(F)$, $v(F')$ is $\text{true}.$

Semantics: Introducing meaning

Let v be a valuation. We extend v to all propositional formulas:

- $v(\top) := \mathrm{true}, v(\bot) := \mathrm{false},$
- $v(\neg F)$ is true iff $v(F)$ is $\mathrm{false}.$
- $v(F \vee F')$ is true iff at least one of $v(F)$, $v(F')$ is $\mathrm{true}.$
- $v(F \wedge F')$ is true iff both, $v(F)$ and $v(F')$ are $\mathrm{true}.$

Semantics: Introducing meaning

Let v be a valuation. We extend v to all propositional formulas:

- $v(\top) := \mathrm{true}, v(\bot) := \mathrm{false}$,
- $v(\neg F)$ is true iff $v(F)$ is false.
- $v(F \vee F')$ is true iff at least one of $v(F)$, $v(F')$ is true.
- $v(F \wedge F')$ is true iff both, $v(F)$ and $v(F')$ are true.

By a structural induction this uniquely defines $v(F)$ for every $F \in \mathbf{P}$.

Semantics: Example

Consider the formula

$$F := \neg P \vee Q$$

Semantics: Example

Consider the formula

$$F := \neg P \vee Q$$

Define a valuation by $v(P) := \text{false}$ and $v(Q) := \text{false}$.

Semantics: Example

Consider the formula

$$F := \neg P \vee Q$$

Define a valuation by $v(P) := \text{false}$ and $v(Q) := \text{false}$. Then

$$v(F) = \text{true}.$$

Satisfiability and tautology

Definition

A formula $F \in \mathbf{P}$ is called *satisfiable* if there exists a valuation v such that $v(F) = \mathrm{true}$.

Satisfiability and tautology

Definition

A formula $F \in \mathbf{P}$ is called *satisfiable* if there exists a valuation v such that $v(F) = \mathrm{true}$.

Definition

A formula $F \in \mathbf{P}$ is called a *tautology* or *valid* if for every valuation v we have $v(F) = \mathrm{true}$.

Satisfiability and tautology

Definition

A formula $F \in \mathbf{P}$ is called *satisfiable* if there exists a valuation v such that $v(F) = \mathrm{true}$.

Definition

A formula $F \in \mathbf{P}$ is called a *tautology* or *valid* if for every valuation v we have $v(F) = \mathrm{true}$.

Exercise: Show that F is satisfiable if and only if $\neg F$ is not valid.

Examples

For each of the following formulas decide satisfiability and validity:

1. $\neg(P \wedge Q) \vee Q \vee R$
2. $\neg P \wedge (Q \vee R \vee P)$
3. $((P \to Q) \to P) \wedge \neg P$

Substitution theorem

Definition

Two formulas $F, F' \in \mathbf{P}$ are called *logically equivalent* if $v(F) = v(F')$ for all valuations v. (Equivalently, if $F \leftrightarrow F'$ is a tautology.) We write $F \equiv F'$.

Substitution theorem

Definition

Two formulas $F, F' \in \mathbf{P}$ are called *logically equivalent* if $v(F) = v(F')$ for all valuations v. (Equivalently, if $F \leftrightarrow F'$ is a tautology.) We write $F \equiv F'$.

For formulas F, X and an atomic proposition P we define $F[P \mapsto X]$ to be the formula where every occurrence of P in F is replaced by X.

Substitution theorem

Definition

Two formulas $F, F' \in \mathbf{P}$ are called *logically equivalent* if $v(F) = v(F')$ for all valuations v. (Equivalently, if $F \leftrightarrow F'$ is a tautology.) We write $F \equiv F'$.

For formulas F, X and an atomic proposition P we define $F[P \mapsto X]$ to be the formula where every occurrence of P in F is replaced by X.

Theorem

Let $X, Y \in \mathbf{P}$ be logically equivalent, $F \in \mathbf{P}$ and P an atomic proposition. Then,

$$v(F[P \mapsto X]) = v(F[P \mapsto Y])$$

for every valuation v.

Definition

Two formulas $F, F' \in \mathbf{P}$ are called *logically equivalent* if $v(F) = v(F')$ for all valuations v. (Equivalently, if $F \leftrightarrow F'$ is a tautology.) We write $F \equiv F'$.

For formulas F, X and an atomic proposition P we define $F[P \mapsto X]$ to be the formula where every occurrence of P in F is replaced by X.

Theorem

Let $X, Y \in \mathbf{P}$ be logically equivalent, $F \in \mathbf{P}$ and P an atomic proposition. Then,

$$v(F[P \mapsto X]) = v(F[P \mapsto Y])$$

for every valuation v.

Together with a list of basic tautologies this enables *simplification* and transformation to *normal forms*.

1 Introduction

2 Syntax and semantics

3 Normal forms

NNF

Definition

A formula is in *negation normal form (NNF)* if the symbol $\neg$ only appears directly in front of literals.

NNF

Definition

A formula is in *negation normal form (NNF)* if the symbol $\neg$ only appears directly in front of literals.

Every propositional formula can be transformed into a logically equivalent formula in NNF.

NNF

Definition

A formula is in *negation normal form (NNF)* if the symbol $\neg$ only appears directly in front of literals.

Every propositional formula can be transformed into a logically equivalent formula in NNF.

Example.

$$\neg(P \wedge \neg Q) \wedge (P \vee R)$$

NNF

Definition

A formula is in *negation normal form (NNF)* if the symbol $\neg$ only appears directly in front of literals.

Every propositional formula can be transformed into a logically equivalent formula in NNF.

Example.

$$\neg(P \wedge \neg Q) \wedge (P \vee R)$$

$$\equiv \neg P \vee Q \wedge (P \vee R)$$

NNF

Definition

A formula is in *negation normal form (NNF)* if the symbol $\neg$ only appears directly in front of literals.

Every propositional formula can be transformed into a logically equivalent formula in NNF.

Example.

$$\neg(P \wedge \neg Q) \wedge (P \vee R)$$

$$\equiv \neg P \vee Q \wedge (P \vee R)$$

DNF and CNF

Definition

A formula is in *disjunctive normal form (DNF)* if it is of the form

$$D_1 \vee D_2 \vee \cdots \vee D_n$$

where each D_i is of the form

$$P_{i1} \wedge \cdots \wedge P_{im_i}$$

with P_{ij} being literals or negated literals.

DNF and CNF

Definition

A formula is in *disjunctive normal form (DNF)* if it is of the form

$$D_1 \vee D_2 \vee \cdots \vee D_n$$

where each D_i is of the form

$$P_{i1} \wedge \cdots \wedge P_{im_i}$$

with P_{ij} being literals or negated literals.

A DNF is "a disjunction of conjunctions", or an "OR of ANDs".

DNF and CNF

Definition

A formula is in *disjunctive normal form (DNF)* if it is of the form

$$D_1 \vee D_2 \vee \cdots \vee D_n$$

where each D_i is of the form

$$P_{i1} \wedge \cdots \wedge P_{im_i}$$

with P_{ij} being literals or negated literals.

A DNF is "a disjunction of conjunctions", or an "OR of ANDs".
Dually, *conjunctive normal form (CNF)* is "a conjunction of disjunctions".
Every formula can be transformed into DNF and CNF.

DNF and CNF

Definition

A formula is in *disjunctive normal form (DNF)* if it is of the form

$$D_1 \vee D_2 \vee \cdots \vee D_n$$

where each D_i is of the form

$$P_{i1} \wedge \cdots \wedge P_{im_i}$$

with P_{ij} being literals or negated literals.

A DNF is "a disjunction of conjunctions", or an "OR of ANDs".
Dually, *conjunctive normal form (CNF)* is "a conjunction of disjunctions".
Every formula can be transformed into DNF and CNF.

DNF and SAT

It is very efficient to check a DNF formula for satisfiability:

A formula in DNF is satisfiable if and only if in at least one of the disjuncts there is no literal that appears negated and unnegated.

Example. The DNF

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

is satisfiable.

Clausal form

Clausal form is the same as CNF.

A *clause* is a disjunction: $P_1 \vee \cdots \vee P_m$ with P_i literals or negated literals. It is often convenient to represent CNF as a list of lists, for example,

$$(P \vee Q \vee \neg R) \wedge P \wedge \neg Q \wedge (R \vee Q)$$

becomes

$$[[P, Q, \neg R], [P], [\neg Q], [R, Q]]$$

To be continued..

Next step: first-order logic!

- FOL satisfiability is only semi-decidable.
- Syntax and semantics will much more involved.
- We will also need more sophisticated (propositional) SAT methods.

References

Melvin Fitting. *First-order Logic and Automated Theorem Proving.* (Springer, 1996)

John Harrison. *Handbook of Practical Logic and Automated Reasoning.* (Cambridge, 2009)