Formalizing first-order logic

Joris Roos

University of Wisconsin-Madison

Sommerakademie Leysin 2018

August 15, 2018

Overview

Introduction

Last time we considered propositional formulas:

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

Introduction

Last time we considered propositional formulas:

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

They consist of constants ($\top$, $\bot$), literals ($A, B, \dots$), logical connectives and punctuation.

Introduction

Last time we considered propositional formulas:

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

They consist of constants ($\top$, $\bot$), literals ($A, B, \dots$), logical connectives and punctuation.

First-order logic (FOL) extends this in two different ways:

- quantifiers: $\forall, \exists$

Introduction

Last time we considered propositional formulas:

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

They consist of constants ($\top$, $\bot$), literals ($A, B, \dots$), logical connectives and punctuation.

First-order logic (FOL) extends this in two different ways:

- quantifiers: $\forall, \exists$
- more complicated atomic formulas, consisting of:

Introduction

Last time we considered propositional formulas:

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

They consist of constants ($\top$, $\bot$), literals ($A, B, \dots$), logical connectives and punctuation.

First-order logic (FOL) extends this in two different ways:

- quantifiers: $\forall, \exists$
- more complicated atomic formulas, consisting of:
    - variables: $x, y, z, \cdots$

Introduction

Last time we considered propositional formulas:

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

They consist of constants ($\top$, $\bot$), literals ($A, B, \dots$), logical connectives and punctuation.

First-order logic (FOL) extends this in two different ways:

- quantifiers: $\forall, \exists$
- more complicated atomic formulas, consisting of:
  - variables: $x, y, z, \cdots$
  - functions: $f(x), g(x, y), ..$

Introduction

Last time we considered propositional formulas:

$$((A \lor D) \to (D \lor \neg B)) \land \neg(A \leftarrow (B \lor C \land D))$$

They consist of constants ($\top$, $\bot$), literals ($A, B, \dots$), logical connectives and punctuation.

First-order logic (FOL) extends this in two different ways:

- quantifiers: $\forall, \exists$
- more complicated atomic formulas, consisting of:
  - variables: $x, y, z, \cdots$
  - functions: $f(x), g(x, y), ..$
  - predicates: $P(x), x = y, \cdots$,

Introduction

Last time we considered propositional formulas:

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

They consist of constants ($\top$, $\bot$), literals ($A, B, \dots$), logical connectives and punctuation.
First-order logic (FOL) extends this in two different ways:

- quantifiers: $\forall, \exists$
- more complicated atomic formulas, consisting of:
  - variables: $x, y, z, \cdots$
  - functions: $f(x), g(x, y), ..$
  - predicates: $P(x), x = y, \cdots$,

First-order logic is powerful enough to formalize "all of mathematics".

Introduction

Last time we considered propositional formulas:

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

They consist of constants ($\top$, $\bot$), literals ($A, B, \dots$), logical connectives and punctuation.

First-order logic (FOL) extends this in two different ways:

- quantifiers: $\forall, \exists$
- more complicated atomic formulas, consisting of:
  - variables: $x, y, z, \cdots$
  - functions: $f(x), g(x, y), ..$
  - predicates: $P(x), x = y, \cdots$,

First-order logic is powerful enough to formalize "all of mathematics".

Example. Continuity

$$\forall x.\forall \varepsilon.\varepsilon > 0 \to (\exists \delta.\delta > 0 \to (\forall y.|x - y| < \delta \to |f(x) - f(y)| < \varepsilon))$$

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.
The alphabet consists of

- Propositional constants: $\top$ (true), $\bot$ (false)

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.
The alphabet consists of

- Propositional constants: $\top$ (true), $\bot$ (false)
- Logical operators: $\neg, \wedge, \vee$

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.
The alphabet consists of

- Propositional constants: $\top$ (true), $\bot$ (false)
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.
The alphabet consists of

- Propositional constants: $\top$ (true), $\bot$ (false)
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,
- Quantifiers: $\forall, \exists$

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.
The alphabet consists of

- Propositional constants: $\top$ (true), $\bot$ (false)
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,
- Quantifiers: $\forall, \exists$
- Function symbols: $f, g, \ldots$

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.
The alphabet consists of

- Propositional constants: $\top$ (true), $\bot$ (false)
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,
- Quantifiers: $\forall, \exists$
- Function symbols: f, g, ...
- Predicate symbols: P, Q, R, ...

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.

The alphabet consists of

- Propositional constants: $\top$ (true), $\bot$ (false)
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,
- Quantifiers: $\forall, \exists$
- Function symbols: f, g, ...
- Predicate symbols: P, Q, R, ...
- Variable symbols: x, y, z, ...

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.
The alphabet consists of

- Propositional constants: $\top$ (true), $\bot$ (false)
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,
- Quantifiers: $\forall, \exists$
- Function symbols: f, g, ...
- Predicate symbols: P, Q, R, ...
- Variable symbols: x, y, z, ...

We will define FOL formulas in three steps:

$$\text{terms} \longrightarrow \text{atomic formulas} \longrightarrow \text{formulas}$$

Admissible symbols

A FOL formula is a string of symbols from a certain alphabet.
The alphabet consists of

- Propositional constants: $\top$ (true), $\bot$ (false)
- Logical operators: $\neg, \wedge, \vee$
- Punctuation: brackets ( ) and ,
- Quantifiers: $\forall, \exists$
- Function symbols: $f, g, \ldots$
- Predicate symbols: $P, Q, R, \ldots$
- Variable symbols: $x, y, z, \ldots$

We will define FOL formulas in three steps:

$$\text{terms} \longrightarrow \text{atomic formulas} \longrightarrow \text{formulas}$$

We often also want to fix the allowed function and predicate symbols (this is called a "language").

FOL language

Definition (Language)

A *first-order language* (or *signature*) L is a pair (F, P) where

FOL language

Definition (Language)

A *first-order language* (or *signature*) L is a pair (F, P) where

- F is a set of pairs (f, n) with f a function symbol and n a natural number.

FOL language

Definition (Language)

A *first-order language* (or *signature*) L is a pair (F, P) where

- F is a set of pairs (f, n) with f a function symbol and n a natural number.
- P is a set of pairs (P, n) with P a predicate symbol and n a natural number.

FOL language

Definition (Language)

A *first-order language* (or *signature*) L is a pair (F, P) where

- F is a set of pairs (f, n) with f a function symbol and n a natural number.
- P is a set of pairs (P, n) with P a predicate symbol and n a natural number.

The number n is called *arity* of the predicate or function symbol. A 0-ary function symbol is also called a constant symbol.

FOL language

Definition (Language)

A *first-order language* (or *signature*) L is a pair (F, P) where

- F is a set of pairs (f, n) with f a function symbol and n a natural number.
- P is a set of pairs (P, n) with P a predicate symbol and n a natural number.

The number n is called *arity* of the predicate or function symbol. A 0-ary function symbol is also called a constant symbol.

Example. The *language of arithmetic* is

$$L_{\mathrm{arith}} = (\{(0,0),(1,0),(+,2),(*,2)\},\{(=,2)\})$$

FOL language

Definition (Language)

A *first-order language* (or *signature*) L is a pair (F, P) where

- F is a set of pairs (f, n) with f a function symbol and n a natural number.
- P is a set of pairs (P, n) with P a predicate symbol and n a natural number.

The number n is called *arity* of the predicate or function symbol. A 0-ary function symbol is also called a constant symbol.

Example. The *language of arithmetic* is

$$L_{\mathrm{arith}} = (\{(0,0),(1,0),(+,2),(*,2)\},\{(=,2)\})$$

The *language of set theory* is

$$L_{\mathrm{set}} = (\{\},\{(=,2),(\in,2)\})$$

Terms

Definition (Term)

The set of *terms* of a FOL language $L = (F, \mathrm{P})$ is the smallest set (of admissible strings) such that

Terms

Definition (Term)

The set of *terms* of a FOL language $L = (F, \mathrm{P})$ is the smallest set (of admissible strings) such that

1. Every variable symbol is a term.

Terms

Definition (Term)

The set of *terms* of a FOL language $L = (F, \mathrm{P})$ is the smallest set (of admissible strings) such that

1. Every variable symbol is a term.
2. If $T_1, \ldots, T_n$ are terms and $(f, n) \in F$, then $f(T_1, \ldots, T_n)$ is a term.

Terms

Definition (Term)

The set of *terms* of a FOL language $L = (F, \mathrm{P})$ is the smallest set (of admissible strings) such that

1. Every variable symbol is a term.
2. If $T_1, \ldots, T_n$ are terms and $(f, n) \in F$, then $f(T_1, \ldots, T_n)$ is a term.

(If $n = 0$ in (2), then the assumption is vacuous, so all constant symbols are terms.)

Terms

Definition (Term)

The set of *terms* of a FOL language $L = (F, \mathrm{P})$ is the smallest set (of admissible strings) such that

1. Every variable symbol is a term.
2. If $T_1, \ldots, T_n$ are terms and $(f, n) \in F$, then $f(T_1, \ldots, T_n)$ is a term.

(If $n = 0$ in (2), then the assumption is vacuous, so all constant symbols are terms.)

This is powerful enough to express all algebraic terms commonly used in mathematics.

Terms

Definition (Term)

The set of *terms* of a FOL language $L = (F, \mathrm{P})$ is the smallest set (of admissible strings) such that

1. Every variable symbol is a term.
2. If $T_1, \ldots, T_n$ are terms and $(f, n) \in F$, then $f(T_1, \ldots, T_n)$ is a term.

(If $n = 0$ in (2), then the assumption is vacuous, so all constant symbols are terms.)

This is powerful enough to express all algebraic terms commonly used in mathematics.

Example.

$$(x + y)(x - y) + y * y$$

is a term of L_{arith} (up to syntax sugar).

Atomic formulas

Definition (Atomic formula)

An *atomic formula* of a FOL language $L = (F, \mathrm{P})$ is a string of the form

$$P(T_1, \ldots, T_n)$$

where $T_1, \ldots, T_n$ are terms of L and $(P, n) \in \mathrm{P}$.

Atomic formulas

Definition (Atomic formula)

An *atomic formula* of a FOL language $L = (F, \mathrm{P})$ is a string of the form

$$P(T_1, \ldots, T_n)$$

where $T_1, \ldots, T_n$ are terms of L and $(P, n) \in \mathrm{P}$.
The propositional constants $\top, \bot$ are also atomic formulas.

(If $n = 0$ then $P(T_1, \ldots, T_n)$ is meant to be just the predicate symbol P.)

Atomic formulas

Definition (Atomic formula)

An *atomic formula* of a FOL language $L = (F, \mathrm{P})$ is a string of the form

$$P(T_1, \ldots, T_n)$$

where $T_1, \ldots, T_n$ are terms of L and $(P, n) \in \mathrm{P}$.
The propositional constants $\top, \bot$ are also atomic formulas.

(If $n = 0$ then $P(T_1, \ldots, T_n)$ is meant to be just the predicate symbol P.)

Example.

$$(x + y)(x - y) + y * y = 0$$

is an atomic formula of L_{arith} (up to syntax sugar).

Atomic formulas

Definition (Atomic formula)

An *atomic formula* of a FOL language $L = (F, \mathrm{P})$ is a string of the form

$$P(T_1, \ldots, T_n)$$

where $T_1, \ldots, T_n$ are terms of L and $(P, n) \in \mathrm{P}$.
The propositional constants $\top, \bot$ are also atomic formulas.

(If $n = 0$ then $P(T_1, \ldots, T_n)$ is meant to be just the predicate symbol P.)

Example.

$$(x + y)(x - y) + y * y = 0$$

is an atomic formula of L_{arith} (up to syntax sugar).

Example.

$$x \in y$$

is an atomic formula of L_{set}.

Formulas

Definition (Formula)

The set of *formulas* of a FOL language L is the smallest set (of strings of admissible symbols) such that

Formulas

Definition (Formula)

The set of *formulas* of a FOL language L is the smallest set (of strings of admissible symbols) such that

1. Atomic formulas of L are formulas.

Formulas

Definition (Formula)

The set of *formulas* of a FOL language L is the smallest set (of strings of admissible symbols) such that

1. Atomic formulas of L are formulas.
2. If F is a formula, then $\neg F$ is a formula.

Formulas

Definition (Formula)

The set of *formulas* of a FOL language L is the smallest set (of strings of admissible symbols) such that

1. Atomic formulas of L are formulas.
2. If F is a formula, then $\neg F$ is a formula.
3. If F, F' are formulas, then $\vee(F, F')$ and $\wedge(F, F')$ are formulas.

Formulas

Definition (Formula)

The set of *formulas* of a FOL language L is the smallest set (of strings of admissible symbols) such that

1. Atomic formulas of L are formulas.
2. If F is a formula, then $\neg F$ is a formula.
3. If F, F' are formulas, then $\vee(F, F')$ and $\wedge(F, F')$ are formulas.
4. If x is a variable symbol and F is a formula, then $\forall(x, F)$ and $\exists(x, F)$ are formulas.

Syntactic sugar

Example. This is a formula (of a suitable FOL language):

$$\exists(x, \forall(y, \wedge(Q(x), \neg P(x, y))))$$

Syntactic sugar

Example. This is a formula (of a suitable FOL language):

$$\exists(x, \forall(y, \wedge(Q(x), \neg P(x, y))))$$

For readability we prefer to write the formula as follows:

$$\exists x. \forall y. Q(x) \wedge \neg P(x, y)$$

Syntactic sugar

Example. This is a formula (of a suitable FOL language):

$$\exists(x, \forall(y, \wedge(Q(x), \neg P(x, y))))$$

For readability we prefer to write the formula as follows:

$$\exists x. \forall y. Q(x) \wedge \neg P(x, y)$$

This is just syntax sugar! We always keep in mind that the formula "actually" looks as above.

Interpretation

To give meaning to a first-order formula we first need to give meaning to its atomic formulas. This requires the following data:

Interpretation

To give meaning to a first-order formula we first need to give meaning to its atomic formulas. This requires the following data:

Definition (Interpretation)

An *interpretation* M of a FOL language $L = (F, \mathrm{P})$ consists of:

Interpretation

To give meaning to a first-order formula we first need to give meaning to its atomic formulas. This requires the following data:

Definition (Interpretation)

An *interpretation* M of a FOL language $L = (F, \mathrm{P})$ consists of:

1. a nonempty set D called the *domain*,

Interpretation

To give meaning to a first-order formula we first need to give meaning to its atomic formulas. This requires the following data:

Definition (Interpretation)

An *interpretation* M of a FOL language $L = (F, \mathrm{P})$ consists of:

1. a nonempty set D called the *domain*,
2. for each n-ary function $(f, n) \in F$ with $n \geq 1$, a function $f_M : D^n \to D$,

Interpretation

To give meaning to a first-order formula we first need to give meaning to its atomic formulas. This requires the following data:

Definition (Interpretation)

An *interpretation* M of a FOL language $L = (F, \mathrm{P})$ consists of:

1. a nonempty set D called the *domain*,
2. for each n-ary function $(f, n) \in F$ with $n \geq 1$, a function $f_M : D^n \to D$,
3. for each n-ary predicate $(P, n) \in \mathrm{P}$, a function $P_M : D^n \to \{\mathrm{true}, \mathrm{false}\}$.

Interpretation

To give meaning to a first-order formula we first need to give meaning to its atomic formulas. This requires the following data:

Definition (Interpretation)

An *interpretation* M of a FOL language $L = (F, \mathrm{P})$ consists of:

1. a nonempty set D called the *domain*,
2. for each n-ary function $(f, n) \in F$ with $n \geq 1$, a function $f_M : D^n \to D$,
3. for each n-ary predicate $(P, n) \in \mathrm{P}$, a function $P_M : D^n \to \{\mathrm{true}, \mathrm{false}\}$.

(In the case $n = 0$, f_M is simply a constant in D; similarly, P_M is a constant truth value.)

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one $M_{\mathbb{N}}$.

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one $M_{\mathbb{N}}$.

Another interpretation of L_{arith} is given by boolean arithmetic:

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one $M_{\mathbb{N}}$.

Another interpretation of L_{arith} is given by boolean arithmetic:

1. $D := \{0, 1\}$

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \text{true}$ iff $x = y$

We call this one $M_{\mathbb{N}}$.

Another interpretation of L_{arith} is given by boolean arithmetic:

1. $D := \{0, 1\}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y \pmod 2$, $*_M(x, y) := x \cdot y \pmod 2)$

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one $M_{\mathbb{N}}$.

Another interpretation of L_{arith} is given by boolean arithmetic:

1. $D := \{0, 1\}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y \pmod 2$, $*_M(x, y) := x \cdot y \pmod 2)$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one M_{bool}.

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one $M_{\mathbb{N}}$.

Another interpretation of L_{arith} is given by boolean arithmetic:

1. $D := \{0, 1\}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y \pmod 2$, $*_M(x, y) := x \cdot y \pmod 2)$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one M_{bool}.

The following is also an interpretation:

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one $M_{\mathbb{N}}$.

Another interpretation of L_{arith} is given by boolean arithmetic:

1. $D := \{0, 1\}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y \pmod 2$, $*_M(x, y) := x \cdot y \pmod 2)$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one M_{bool}.

The following is also an interpretation:

1. $D := \mathbb{Z}$,

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one $M_{\mathbb{N}}$.

Another interpretation of L_{arith} is given by boolean arithmetic:

1. $D := \{0, 1\}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y \pmod 2$, $*_M(x, y) := x \cdot y \pmod 2)$
3. $=_M (x, y) := \mathrm{true}$ iff $x = y$

We call this one M_{bool}.

The following is also an interpretation:

1. $D := \mathbb{Z}$,
2. $0_M := 7$, $1_M := 0$, $+_M(x, y) := x - 3y^2$, $*_M(x, y) := x + y$,

Examples

An interpretation of L_{arith} is given by arithmetic of natural numbers:

1. $D := \mathbb{N}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y$, $*_M(x, y) := x * y$
3. $=_M (x, y) := \text{true iff } x = y$

We call this one $M_{\mathbb{N}}$.

Another interpretation of L_{arith} is given by boolean arithmetic:

1. $D := \{0, 1\}$
2. $0_M := 0$, $1_M := 1$, $+_M(x, y) := x + y \pmod 2$, $*_M(x, y) := x \cdot y \pmod 2)$
3. $=_M (x, y) := \text{true iff } x = y$

We call this one M_{bool}.

The following is also an interpretation:

1. $D := \mathbb{Z}$,
2. $0_M := 7$, $1_M := 0$, $+_M(x, y) := x - 3y^2$, $*_M(x, y) := x + y$,
3. $=_M (x, y) := \text{true iff } x \neq y$

Term evaluation

Definition

A *valuation* v for an interpretation M is a map from the set of variable symbols to the domain D: $v(x) \in D$ for every variable symbol x.

Term evaluation

Definition

A *valuation* v for an interpretation M is a map from the set of variable symbols to the domain D: $v(x) \in D$ for every variable symbol x.

Let M be an interpretation and v a valuation. Then to each term t we assign a value $\mathrm{ev}_{M,v}(t) \in D$ such that

Term evaluation

Definition

A *valuation* v for an interpretation M is a map from the set of variable symbols to the domain D: $v(x) \in D$ for every variable symbol x.

Let M be an interpretation and v a valuation. Then to each term t we assign a value $\mathrm{ev}_{M,v}(t) \in D$ such that

1. $\mathrm{ev}_{M,v}(x) = v(x)$ if x is a variable symbol,

Term evaluation

Definition

A *valuation* v for an interpretation M is a map from the set of variable symbols to the domain D: $v(x) \in D$ for every variable symbol x.

Let M be an interpretation and v a valuation. Then to each term t we assign a value $\mathrm{ev}_{M,v}(t) \in D$ such that

1. $\mathrm{ev}_{M,v}(x) = v(x)$ if x is a variable symbol,
2. $$\mathrm{ev}_{M,v}(f(t_1, \dots, t_n)) = f_M(\mathrm{ev}_{M,v}(t_1), \dots, \mathrm{ev}_{M,v}(t_n))$$

   for every n-ary function symbol f.

Term evaluation

Definition

A *valuation* v for an interpretation M is a map from the set of variable symbols to the domain D: $v(x) \in D$ for every variable symbol x.

Let M be an interpretation and v a valuation. Then to each term t we assign a value $\mathrm{ev}_{M,v}(t) \in D$ such that

1. $\mathrm{ev}_{M,v}(x) = v(x)$ if x is a variable symbol,
2. $$\mathrm{ev}_{M,v}(f(t_1, \ldots, t_n)) = f_M(\mathrm{ev}_{M,v}(t_1), \ldots, \mathrm{ev}_{M,v}(t_n))$$

   for every n-ary function symbol f.

By structural induction this defines a unique map from the set of terms to the domain D.

Evaluation of formulas

For each formula F, interpretation M and valuation v we want to define a meaning $\mathrm{ev}_{M,v}(F) \in \{\mathrm{true}, \mathrm{false}\}$.

Evaluation of formulas

For each formula F, interpretation M and valuation v we want to define a meaning $\mathrm{ev}_{M,v}(F) \in \{\mathrm{true}, \mathrm{false}\}$.
The idea is again structural induction, but there is a catch!

Evaluation of formulas

For each formula F, interpretation M and valuation v we want to define a meaning $\mathrm{ev}_{M,v}(F) \in \{\mathrm{true}, \mathrm{false}\}$.
The idea is again structural induction, but there is a catch!
The semantics of the quantifiers require us to modify the assignment v for the variables bound by a quantifier.

Evaluation of formulas

For each formula F, interpretation M and valuation v we want to define a meaning $\mathrm{ev}_{M,v}(F) \in \{\mathrm{true}, \mathrm{false}\}$.
The idea is again structural induction, but there is a catch!
The semantics of the quantifiers require us to modify the assignment v for the variables bound by a quantifier.
For every variable symbol x and every $a \in D$ we define

$$v[x \mapsto a]$$

to be the assignment that maps x to a and every $y \neq x$ to $v(y)$.

Evaluation of formulas

For each formula F, interpretation M and valuation v we want to define a meaning $\mathrm{ev}_{M,v}(F) \in \{\mathrm{true}, \mathrm{false}\}$.
The idea is again structural induction, but there is a catch!
The semantics of the quantifiers require us to modify the assignment v for the variables bound by a quantifier.
For every variable symbol x and every $a \in D$ we define

$$v[x \mapsto a]$$

to be the assignment that maps x to a and every $y \neq x$ to $v(y)$.

Evaluation of formulas

Now we can run the structural induction:

1. $\mathrm{ev}_{M,v}(P(t_1, \dots, t_n)) := P_M(\mathrm{ev}_{M,v}(t_1), \dots, \mathrm{ev}_{M,v}(t_n))$
   for every n-ary predicate symbol P.

Evaluation of formulas

Now we can run the structural induction:

1. $\mathrm{ev}_{M,v}(P(t_1, \dots, t_n)) := P_M(\mathrm{ev}_{M,v}(t_1), \dots, \mathrm{ev}_{M,v}(t_n))$ for every n-ary predicate symbol P.
2. $\mathrm{ev}_{M,v}(\neg F)$ is true iff $\mathrm{ev}_{M,v}(F)$ is false.

Evaluation of formulas

Now we can run the structural induction:

1. $\mathrm{ev}_{M,v}(P(t_1,\ldots,t_n)) := P_M(\mathrm{ev}_{M,v}(t_1),\ldots,\mathrm{ev}_{M,v}(t_n))$ for every n-ary predicate symbol P.
2. $\mathrm{ev}_{M,v}(\neg F)$ is true iff $\mathrm{ev}_{M,v}(F)$ is false.
3. $\mathrm{ev}_{M,v}(F \vee F')$ is true iff at least one of $\mathrm{ev}_{M,v}(F), \mathrm{ev}_{M,v}(F')$ is true.

Evaluation of formulas

Now we can run the structural induction:

1. $\mathrm{ev}_{M,v}(P(t_1, \ldots, t_n)) := P_M(\mathrm{ev}_{M,v}(t_1), \ldots, \mathrm{ev}_{M,v}(t_n))$ for every n-ary predicate symbol P.
2. $\mathrm{ev}_{M,v}(\neg F)$ is true iff $\mathrm{ev}_{M,v}(F)$ is false.
3. $\mathrm{ev}_{M,v}(F \vee F')$ is true iff at least one of $\mathrm{ev}_{M,v}(F), \mathrm{ev}_{M,v}(F')$ is true.
4. $\mathrm{ev}_{M,v}(F \wedge F')$ is true iff both, $\mathrm{ev}_{M,v}(F)$ and $\mathrm{ev}_{M,v}(F')$ are true.

Evaluation of formulas

Now we can run the structural induction:

1. $\mathrm{ev}_{M,v}(P(t_1,\ldots,t_n)) := P_M(\mathrm{ev}_{M,v}(t_1),\ldots,\mathrm{ev}_{M,v}(t_n))$ for every n-ary predicate symbol P.
2. $\mathrm{ev}_{M,v}(\neg F)$ is true iff $\mathrm{ev}_{M,v}(F)$ is false.
3. $\mathrm{ev}_{M,v}(F \vee F')$ is true iff at least one of $\mathrm{ev}_{M,v}(F), \mathrm{ev}_{M,v}(F')$ is true.
4. $\mathrm{ev}_{M,v}(F \wedge F')$ is true iff both, $\mathrm{ev}_{M,v}(F)$ and $\mathrm{ev}_{M,v}(F')$ are true.
5. $\mathrm{ev}_{M,v}(\exists x.F)$ is true iff there exists $a \in D$ such that $\mathrm{ev}_{M,v[x\mapsto a]}(F)$ is true.

Evaluation of formulas

Now we can run the structural induction:

1. $\mathrm{ev}_{M,v}(P(t_1, \ldots, t_n)) := P_M(\mathrm{ev}_{M,v}(t_1), \ldots, \mathrm{ev}_{M,v}(t_n))$ for every n-ary predicate symbol P.
2. $\mathrm{ev}_{M,v}(\neg F)$ is true iff $\mathrm{ev}_{M,v}(F)$ is false.
3. $\mathrm{ev}_{M,v}(F \vee F')$ is true iff at least one of $\mathrm{ev}_{M,v}(F), \mathrm{ev}_{M,v}(F')$ is true.
4. $\mathrm{ev}_{M,v}(F \wedge F')$ is true iff both, $\mathrm{ev}_{M,v}(F)$ and $\mathrm{ev}_{M,v}(F')$ are true.
5. $\mathrm{ev}_{M,v}(\exists x.F)$ is true iff there exists $a \in D$ such that $\mathrm{ev}_{M,v[x \mapsto a]}(F)$ is true.
6. $\mathrm{ev}_{M,v}(\forall x.F)$ is true iff for all $a \in D$, $\mathrm{ev}_{M,v[x \mapsto a]}(F)$ is true.

Evaluation of formulas

Now we can run the structural induction:

1. $\mathrm{ev}_{M,v}(P(t_1,\ldots,t_n)) := P_M(\mathrm{ev}_{M,v}(t_1),\ldots,\mathrm{ev}_{M,v}(t_n))$ for every n-ary predicate symbol P.
2. $\mathrm{ev}_{M,v}(\neg F)$ is true iff $\mathrm{ev}_{M,v}(F)$ is false.
3. $\mathrm{ev}_{M,v}(F \vee F')$ is true iff at least one of $\mathrm{ev}_{M,v}(F), \mathrm{ev}_{M,v}(F')$ is true.
4. $\mathrm{ev}_{M,v}(F \wedge F')$ is true iff both, $\mathrm{ev}_{M,v}(F)$ and $\mathrm{ev}_{M,v}(F')$ are true.
5. $\mathrm{ev}_{M,v}(\exists x.F)$ is true iff there exists $a \in D$ such that $\mathrm{ev}_{M,v[x\mapsto a]}(F)$ is true.
6. $\mathrm{ev}_{M,v}(\forall x.F)$ is true iff for all $a \in D$, $\mathrm{ev}_{M,v[x\mapsto a]}(F)$ is true.

Beware: the domain D is nonempty, but otherwise arbitrary (e.g. could be uncountably infinite) and the functions f_M, P_M need not be computable.

Evaluation of formulas

Now we can run the structural induction:

1. $\mathrm{ev}_{M,v}(P(t_1, \ldots, t_n)) := P_M(\mathrm{ev}_{M,v}(t_1), \ldots, \mathrm{ev}_{M,v}(t_n))$ for every n-ary predicate symbol P.
2. $\mathrm{ev}_{M,v}(\neg F)$ is true iff $\mathrm{ev}_{M,v}(F)$ is false.
3. $\mathrm{ev}_{M,v}(F \vee F')$ is true iff at least one of $\mathrm{ev}_{M,v}(F), \mathrm{ev}_{M,v}(F')$ is true.
4. $\mathrm{ev}_{M,v}(F \wedge F')$ is true iff both, $\mathrm{ev}_{M,v}(F)$ and $\mathrm{ev}_{M,v}(F')$ are true.
5. $\mathrm{ev}_{M,v}(\exists x.F)$ is true iff there exists $a \in D$ such that $\mathrm{ev}_{M,v[x\mapsto a]}(F)$ is true.
6. $\mathrm{ev}_{M,v}(\forall x.F)$ is true iff for all $a \in D$, $\mathrm{ev}_{M,v[x\mapsto a]}(F)$ is true.

Beware: the domain D is nonempty, but otherwise arbitrary (e.g. could be uncountably infinite) and the functions f_M, P_M need not be computable. To enable a straightforward computer implementation of the semantics we need to restrict to finite D.

Evaluation of formulas

Now we can run the structural induction:

1. $\mathrm{ev}_{M,v}(P(t_1,\ldots,t_n)) := P_M(\mathrm{ev}_{M,v}(t_1),\ldots,\mathrm{ev}_{M,v}(t_n))$ for every n-ary predicate symbol P.
2. $\mathrm{ev}_{M,v}(\neg F)$ is true iff $\mathrm{ev}_{M,v}(F)$ is false.
3. $\mathrm{ev}_{M,v}(F \vee F')$ is true iff at least one of $\mathrm{ev}_{M,v}(F), \mathrm{ev}_{M,v}(F')$ is true.
4. $\mathrm{ev}_{M,v}(F \wedge F')$ is true iff both, $\mathrm{ev}_{M,v}(F)$ and $\mathrm{ev}_{M,v}(F')$ are true.
5. $\mathrm{ev}_{M,v}(\exists x.F)$ is true iff there exists $a \in D$ such that $\mathrm{ev}_{M,v[x\mapsto a]}(F)$ is true.
6. $\mathrm{ev}_{M,v}(\forall x.F)$ is true iff for all $a \in D$, $\mathrm{ev}_{M,v[x\mapsto a]}(F)$ is true.

Beware: the domain D is nonempty, but otherwise arbitrary (e.g. could be uncountably infinite) and the functions f_M, P_M need not be computable. To enable a straightforward computer implementation of the semantics we need to restrict to finite D.

Free variables

Definition

The set of *free variables* of a formula F is defined as the set of variable symbols that are not bound by quantifiers and denoted $FV(F)$.

Free variables

Definition

The set of *free variables* of a formula F is defined as the set of variable symbols that are not bound by quantifiers and denoted $FV(F)$.
A formula without free variables is called a *sentence*.

Free variables

Definition

The set of *free variables* of a formula F is defined as the set of variable symbols that are not bound by quantifiers and denoted $FV(F)$.
A formula without free variables is called a *sentence*.

Our setup allows to prove properties of the semantics:

Free variables

Definition

The set of *free variables* of a formula F is defined as the set of variable symbols that are not bound by quantifiers and denoted $FV(F)$.
A formula without free variables is called a *sentence*.

Our setup allows to prove properties of the semantics:

Lemma

Let F be a formula and M an interpretation. If v, v' are valuations such that $v(x) = v'(x)$ for every $x \in FV(F)$, then

$$\mathrm{ev}_{M,v}(F) = \mathrm{ev}_{M,v'}(F).$$

Free variables

Definition

The set of *free variables* of a formula F is defined as the set of variable symbols that are not bound by quantifiers and denoted $FV(F)$.
A formula without free variables is called a *sentence*.

Our setup allows to prove properties of the semantics:

Lemma

Let F be a formula and M an interpretation. If v, v' are valuations such that $v(x) = v'(x)$ for every $x \in FV(F)$, then

$$\mathrm{ev}_{M,v}(F) = \mathrm{ev}_{M,v'}(F).$$

As a corollary, if F is a sentence, then $\mathrm{ev}_{M,v}(F)$ does not depend on the valuation v.

Models

Definition (Model)

Let F be a formula. An interpretation M is called a *model of F* if $\mathrm{ev}_{M,v}(F) = \mathrm{true}$ for every valuation v (we also say that F *holds*).

Models

Definition (Model)

Let F be a formula. An interpretation M is called a *model of F* if $\mathrm{ev}_{M,v}(F) = \mathrm{true}$ for every valuation v (we also say that F *holds*).

Definition

A formula F is called *satisfiable* if it has a model.

Models

Definition (Model)

Let F be a formula. An interpretation M is called a *model of F* if $\mathrm{ev}_{M,v}(F) = \mathrm{true}$ for every valuation v (we also say that F *holds*).

Definition

A formula F is called *satisfiable* if it has a model.

Definition

A formula F is called *logically valid* or a *tautology* if every interpretation is a model.

Models

Definition (Model)

Let F be a formula. An interpretation M is called a *model of F* if $\mathrm{ev}_{M,v}(F) = \mathrm{true}$ for every valuation v (we also say that F *holds*).

Definition

A formula F is called *satisfiable* if it has a model.

Definition

A formula F is called *logically valid* or a *tautology* if every interpretation is a model.

- Fix a language L, fix a formula F.

- Fix a language L, fix a formula F.
- We would like to describe a procedure that decides if F is satisfiable, in finitely many steps.

- Fix a language L, fix a formula F.
- We would like to describe a procedure that decides if F is satisfiable, in finitely many steps.
- Unfortunately, this is generally not possible (first-order logic is "undecidable" if L is non-trivial enough).

- Fix a language L, fix a formula F.
- We would like to describe a procedure that decides if F is satisfiable, in finitely many steps.
- Unfortunately, this is generally not possible (first-order logic is "undecidable" if L is non-trivial enough).
- However, it is "semi-decidable": if F is not satisfiable, then we can verify that in finite time.

- Fix a language L, fix a formula F.
- We would like to describe a procedure that decides if F is satisfiable, in finitely many steps.
- Unfortunately, this is generally not possible (first-order logic is "undecidable" if L is non-trivial enough).
- However, it is "semi-decidable": if F is not satisfiable, then we can verify that in finite time.

Reduction to propositional logic

- The basic idea is to reduce the problem to solving propositional SAT.

Reduction to propositional logic

- The basic idea is to reduce the problem to solving propositional SAT.
- To do that we first “get rid” of quantifiers.

Reduction to propositional logic

- The basic idea is to reduce the problem to solving propositional SAT.
- To do that we first “get rid” of quantifiers.
- Each transformation will preserve satisfiability, but not necessarily logic equivalence.

Step 0: Remove free variables

Given any formula F with free variables $FV(F) = \{x_1, \dots, x_n\}$ we can form the *generalization* $\mathrm{gen}(F)$:

Step 0: Remove free variables

Given any formula F with free variables $FV(F) = \{x_1, \ldots, x_n\}$ we can form the *generalization* $\mathrm{gen}(F)$:

$$\forall x_1. \cdots \forall x_n. F$$

Step 0: Remove free variables

Given any formula F with free variables $FV(F) = \{x_1, \ldots, x_n\}$ we can form the *generalization* $\mathrm{gen}(F)$:

$$\forall x_1. \cdots \forall x_n. F$$

(This can be made precise by an inductive definition.)
F and $\mathrm{gen}(F)$ are not logically equivalent, but *equisatisfiable*:

Step 0: Remove free variables

Given any formula F with free variables $FV(F) = \{x_1, \ldots, x_n\}$ we can form the *generalization* $\mathrm{gen}(F)$:

$$\forall x_1. \cdots \forall x_n. F$$

(This can be made precise by an inductive definition.)
F and $\mathrm{gen}(F)$ are not logically equivalent, but *equisatisfiable*:
F is satisfiable if and only if $\mathrm{gen}(F)$ is satisfiable.

Step 0: Remove free variables

Given any formula F with free variables $FV(F) = \{x_1, \ldots, x_n\}$ we can form the *generalization* $\mathrm{gen}(F)$:

$$\forall x_1. \cdots \forall x_n. F$$

(This can be made precise by an inductive definition.)
F and $\mathrm{gen}(F)$ are not logically equivalent, but *equisatisfiable*:
F is satisfiable if and only if $\mathrm{gen}(F)$ is satisfiable.
Thus, from now on we assume that F is a sentence.

Step 1: Convert to prenex normal form

A formula is in *prenex normal form (PNF)* if all the quantifiers appear on the outside.

Step 1: Convert to prenex normal form

A formula is in *prenex normal form (PNF)* if all the quantifiers appear on the outside.

Example.

$$\forall x.\forall y.\exists z.(P(x) \wedge Q(y,z))$$

is in PNF.

Step 1: Convert to prenex normal form

A formula is in *prenex normal form (PNF)* if all the quantifiers appear on the outside.
Example.

$$\forall x.\forall y.\exists z.(P(x) \wedge Q(y, z))$$

is in PNF.

$$(\exists x.P(x)) \rightarrow \forall y.P(y)$$

is not in PNF.

Step 1: Convert to prenex normal form

A formula is in *prenex normal form (PNF)* if all the quantifiers appear on the outside.

Example.

$$\forall x.\forall y.\exists z.(P(x) \wedge Q(y,z))$$

is in PNF.

$$(\exists x.P(x)) \to \forall y.P(y)$$

is not in PNF.

Every formula can be transformed into a logically equivalent formula in PNF.

Step 2: Skolemization

The main idea is that the following are equivalent:

$$\forall x \in D \exists y \in D \text{ s.t. } P(x,y) \text{ holds}$$

Step 2: Skolemization

The main idea is that the following are equivalent:

$$\forall x \in D \exists y \in D \text{ s.t. } P(x, y) \text{ holds}$$

$$\exists f : D \to D \text{ s.t. } \forall x \in D \text{ we have } P(x, f(x))$$

Step 2: Skolemization

The main idea is that the following are equivalent:

$$\forall x \in D \exists y \in D \text{ s.t. } P(x, y) \text{ holds}$$

$$\exists f : D \to D \text{ s.t. } \forall x \in D \text{ we have } P(x, f(x))$$

(In general, this is essentially the axiom of choice, but it suffices to consider countable D here, which does not require choice.)

Step 2: Skolemization

The main idea is that the following are equivalent:

$$\forall x \in D \exists y \in D \text{ s.t. } P(x, y) \text{ holds}$$

$$\exists f : D \to D \text{ s.t. } \forall x \in D \text{ we have } P(x, f(x))$$

(In general, this is essentially the axiom of choice, but it suffices to consider countable D here, which does not require choice.)

But we cannot transform formulas of the form

$$\forall x . \exists y . P(x, y)$$

Step 2: Skolemization

The main idea is that the following are equivalent:

$$\forall x \in D \exists y \in D \text{ s.t. } P(x, y) \text{ holds}$$

$$\exists f : D \to D \text{ s.t. } \forall x \in D \text{ we have } P(x, f(x))$$

(In general, this is essentially the axiom of choice, but it suffices to consider countable D here, which does not require choice.)

But we cannot transform formulas of the form

$$\forall x. \exists y. P(x, y)$$

into

$$\exists f. \forall x. P(x, f(x)),$$

Step 2: Skolemization

The main idea is that the following are equivalent:

$$\forall x \in D \exists y \in D \text{ s.t. } P(x, y) \text{ holds}$$

$$\exists f : D \to D \text{ s.t. } \forall x \in D \text{ we have } P(x, f(x))$$

(In general, this is essentially the axiom of choice, but it suffices to consider countable D here, which does not require choice.)

But we cannot transform formulas of the form

$$\forall x. \exists y. P(x, y)$$

into

$$\exists f. \forall x. P(x, f(x)),$$

because the latter is not a first-order formula (because f is not a variable symbol).

Step 2: Skolemization

The main idea is that the following are equivalent:

$$\forall x \in D \exists y \in D \text{ s.t. } P(x, y) \text{ holds}$$

$$\exists f : D \to D \text{ s.t. } \forall x \in D \text{ we have } P(x, f(x))$$

(In general, this is essentially the axiom of choice, but it suffices to consider countable D here, which does not require choice.)

But we cannot transform formulas of the form

$$\forall x . \exists y . P(x, y)$$

into

$$\exists f . \forall x . P(x, f(x)),$$

because the latter is not a first-order formula (because f is not a variable symbol).

Step 2: Skolemization

Instead, we transform

$$\forall x. \exists y. P(x, y)$$

Step 2: Skolemization

Instead, we transform

$$\forall x.\exists y.P(x,y)$$

into

$$\forall x.P(x,f(x))$$

Step 2: Skolemization

Instead, we transform

$$\forall x. \exists y. P(x, y)$$

into

$$\forall x. P(x, f(x))$$

and augment the language L by a new function symbol f (called a *Skolem function*).

Step 2: Skolemization

Instead, we transform

$$\forall x. \exists y. P(x, y)$$

into

$$\forall x. P(x, f(x))$$

and augment the language L by a new function symbol f (called a *Skolem function*).

If F is in prenex normal form, we can iterate this to produce a new equisatisfiable formula $\mathrm{skolemize}(F)$ that has no existential quantifiers.

Step 2: Skolemization

Instead, we transform

$$\forall x.\exists y.P(x,y)$$

into

$$\forall x.P(x,f(x))$$

and augment the language L by a new function symbol f (called a *Skolem function*).

If F is in prenex normal form, we can iterate this to produce a new equisatisfiable formula $\mathrm{skolemize}(F)$ that has no existential quantifiers.

Example.

$$\mathrm{skolemize}(\exists x.\forall y.\exists z.\forall u.\exists v.P(x,y,z,u,v))$$

Step 2: Skolemization

Instead, we transform

$$\forall x.\exists y.P(x,y)$$

into

$$\forall x.P(x,f(x))$$

and augment the language L by a new function symbol f (called a *Skolem function*).
If F is in prenex normal form, we can iterate this to produce a new equisatisfiable formula $\mathrm{skolemize}(F)$ that has no existential quantifiers.
Example.

$$\mathrm{skolemize}(\exists x.\forall y.\exists z.\forall u.\exists v.P(x,y,z,u,v))$$

$$= \forall y.\forall u.P(c,y,f(y),u,g(y,u))$$

(c, f, g are new function symbols)

Step 3: Remove universal quantifiers

We are now left with a formula of the form

$$\forall x_1. \cdots \forall x_n. F'$$

where F' contains no quantifiers.

Step 3: Remove universal quantifiers

We are now left with a formula of the form

$$\forall x_1. \cdots \forall x_n. F'$$

where F' contains no quantifiers.
By definition of satisfiability, we can simply remove the quantifiers: the quantifier-free formula

$$F'$$

is equisatisfiable to the previous, and thus equisatisfiable to the original formula F.

Step 4: Iterate through ground instances

We are now left with deciding whether a quantifier-free formula is satisfiable.

Step 4: Iterate through ground instances

We are now left with deciding whether a quantifier-free formula is satisfiable.

Theorem (Herbrand compactness theorem)

A quantifier-free formula F is satisfiable if and only if every finite set of ground instances is satisfiable.

Step 4: Iterate through ground instances

We are now left with deciding whether a quantifier-free formula is satisfiable.

Theorem (Herbrand compactness theorem)

A quantifier-free formula F is satisfiable if and only if every finite set of ground instances is satisfiable.

A *ground term* is one that only consists of function symbols (including constant symbols).

Step 4: Iterate through ground instances

We are now left with deciding whether a quantifier-free formula is satisfiable.

Theorem (Herbrand compactness theorem)

A quantifier-free formula F is satisfiable if and only if every finite set of ground instances is satisfiable.

A *ground term* is one that only consists of function symbols (including constant symbols).
A *ground instance* is the propositional formula that arises from replacing each of the free variables of F by a ground term and interpreting atomic formulas as propositional literals.

Step 4: Iterate through ground instances

We are now left with deciding whether a quantifier-free formula is satisfiable.

Theorem (Herbrand compactness theorem)

A quantifier-free formula F is satisfiable if and only if every finite set of ground instances is satisfiable.

A *ground term* is one that only consists of function symbols (including constant symbols).
A *ground instance* is the propositional formula that arises from replacing each of the free variables of F by a ground term and interpreting atomic formulas as propositional literals.
This is in principle an automatic theorem prover!

Step 4: Iterate through ground instances

We are now left with deciding whether a quantifier-free formula is satisfiable.

Theorem (Herbrand compactness theorem)

A quantifier-free formula F is satisfiable if and only if every finite set of ground instances is satisfiable.

A *ground term* is one that only consists of function symbols (including constant symbols).
A *ground instance* is the propositional formula that arises from replacing each of the free variables of F by a ground term and interpreting atomic formulas as propositional literals.
This is in principle an automatic theorem prover!
It is guaranteed to terminate in the case that the original formula is not satisfiable.

Example

Say we want to prove that the *drinker's principle* holds:

$$\exists x.\forall y.(P(x) \to P(y))$$

(the language contains only the unary predicate P)

Example

Say we want to prove that the *drinker's principle* holds:

$$\exists x.\forall y.(P(x) \to P(y))$$

(the language contains only the unary predicate P)
Equivalently we can show that the negation is not satisfiable:

$$\neg(\exists x.\forall y.(P(x) \to P(y)))$$

Example

Say we want to prove that the *drinker's principle* holds:

$$\exists x.\forall y.(P(x) \to P(y))$$

(the language contains only the unary predicate P)
Equivalently we can show that the negation is not satisfiable:

$$\neg(\exists x.\forall y.(P(x) \to P(y)))$$

Step 1: convert to PNF

$$\forall x.\exists y.\neg(P(x) \to P(y))$$

Example

Say we want to prove that the *drinker's principle* holds:

$$\exists x.\forall y.(P(x) \to P(y))$$

(the language contains only the unary predicate P)
Equivalently we can show that the negation is not satisfiable:

$$\neg(\exists x.\forall y.(P(x) \to P(y)))$$

Step 1: convert to PNF

$$\forall x.\exists y.\neg(P(x) \to P(y))$$

Step 2: Skolemize

$$\forall x.\neg(P(x) \to P(f(x))$$

(the language no contains the unary predicate P and the unary function f)

Example

Say we want to prove that the *drinker's principle* holds:

$$\exists x.\forall y.(P(x) \to P(y))$$

(the language contains only the unary predicate P)
Equivalently we can show that the negation is not satisfiable:

$$\neg(\exists x.\forall y.(P(x) \to P(y)))$$

Step 1: convert to PNF

$$\forall x.\exists y.\neg(P(x) \to P(y))$$

Step 2: Skolemize

$$\forall x.\neg(P(x) \to P(f(x))$$

(the language no contains the unary predicate P and the unary function f)

Step 3: Remove quantifiers

$$\neg(P(x) \to P(f(x)))$$

$\neg(P(x) \to P(f(x)))$

Step 3': For convenience, let us bring the formula into DNF

$$P(x) \land \neg P(f(x))$$

$\neg(P(x) \to P(f(x)))$

Step 3': For convenience, let us bring the formula into DNF

$$P(x) \wedge \neg P(f(x))$$

Step 4: Iterate through ground instances

$\neg(P(x) \to P(f(x)))$

Step 3': For convenience, let us bring the formula into DNF

$$P(x) \wedge \neg P(f(x))$$

Step 4: Iterate through ground instances

(for the set of ground terms to be non-empty we need to add a constant symbol c to the language, but this does not change satisfiability)

$\neg(P(x) \to P(f(x)))$

Step 3': For convenience, let us bring the formula into DNF

$$P(x) \wedge \neg P(f(x))$$

Step 4: Iterate through ground instances

(for the set of ground terms to be non-empty we need to add a constant symbol c to the language, but this does not change satisfiability)

1. First ground term $x = c$:

$$P(c) \wedge \neg P(f(c))$$

$\neg(P(x) \to P(f(x)))$

Step 3': For convenience, let us bring the formula into DNF

$$P(x) \wedge \neg P(f(x))$$

Step 4: Iterate through ground instances

(for the set of ground terms to be non-empty we need to add a constant symbol c to the language, but this does not change satisfiability)

1. First ground term $x = c$:

$$P(c) \wedge \neg P(f(c))$$

This is a single propositional formula with literals $P(c)$ and $P(f(c))$!

$\neg(P(x) \to P(f(x)))$

Step 3': For convenience, let us bring the formula into DNF

$$P(x) \land \neg P(f(x))$$

Step 4: Iterate through ground instances

(for the set of ground terms to be non-empty we need to add a constant symbol c to the language, but this does not change satisfiability)

1. First ground term $x = c$:

$$P(c) \land \neg P(f(c))$$

This is a single propositional formula with literals $P(c)$ and $P(f(c))$!
This is still satisfiable.

$\neg(P(x) \to P(f(x)))$

Step 3': For convenience, let us bring the formula into DNF

$$P(x) \land \neg P(f(x))$$

Step 4: Iterate through ground instances

(for the set of ground terms to be non-empty we need to add a constant symbol c to the language, but this does not change satisfiability)

1. First ground term $x = c$:

$$P(c) \land \neg P(f(c))$$

This is a single propositional formula with literals $P(c)$ and $P(f(c))$!
This is still satisfiable.

2. Add second ground term $x = f(c)$:

$$P(c) \land \neg P(f(c)), P(f(c)) \land \neg P(f(f(c)))\}$$

These are not simultanously satisfiable! QED

References

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

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