First-Order Predicate Logic (2)

Predicate Logic (2)

- Understanding first-order predicate logic formulas.
- Satisfiability and undecidability of satisfiability.
- Tautology, logical consequence, and logical equivalence.
- Completeness of first-order predicate logic
- Incompleteness of arithmetic and second-order logic.

Translations into Predicate Logic

- “Every house is a physical object“ is translated as

$$\forall x.(\mathbf{house}(x) \to \mathbf{physical\_object}(x)),$$

  where **house** and **physical_object** are unary predicate symbols.

- “Some physical objects are houses“' is translated as

$$\exists x.(\mathbf{physical\_object}(x) \land \mathbf{house}(x))$$

- ”Every house has an owner“ or, equivalently, “every house is owned by somebody“ is translated as

$$\forall x(\mathbf{house}(x) \to \exists y.\mathbf{owns}(y, x)),$$

  here **owns** is a binary predicate symbol.

- ”Everybody owns a house“ is translated as

$$\forall x.\exists y.(\mathbf{owns}(x, y) \land \mathbf{house}(y))$$

Translations into Predicate Logic

- "Sue owns a house" is translated as

$$\exists x.(\textbf{owns}(\textbf{Sue}, x) \wedge \textbf{house}(x))$$

  where **Sue** is an individual constant symbol.

- "Peter does not own a house" is translated as

$$\neg\exists x.(\textbf{owns}(\textbf{Peter}, x) \wedge \textbf{house}(x))$$

- "Somebody does not own a house" is translated as

$$\exists x.\forall y.(\textbf{owns}(x, y) \rightarrow \neg\textbf{house}(y))$$

The truth relation

Let S be the signature consisting of the unary predicates **house** and **human**, the binary predicate **owns**, and the individual constant **Sue**. Give an S-interpretation $\mathcal{F}$ with $\mathcal{F} \models G$ for the following sentences G:

- there are houses: $\exists x.\mathbf{house}(x)$
- there are human beings: $\exists x.\mathbf{human}(x)$
- no house is a human being: $\forall x.(\mathbf{house}(x) \to \neg\mathbf{human}(x))$
- some humans own a house: $\exists x.\exists y.(\mathbf{human}(x) \land \mathbf{house}(y) \land \mathbf{owns}(x, y))$
- Sue is human: $\mathbf{human}(\mathbf{Sue})$
- Sue does not own a house: $\neg\exists x.(\mathbf{owns}(\mathbf{Sue}, x) \land \mathbf{house}(x))$
- every house has an owner: $\forall x.(\mathbf{house}(x) \to \exists y.\mathbf{owns}(y, x))$

The S-interpretation $\mathcal{F}$

We can take, for example, $\mathcal{F}$ defined by

- $D^{\mathcal{F}} = \{a, b, c, d, e\}$;
- **human**$^{\mathcal{F}} = \{a, b, c\}$;
- **house**$^{\mathcal{F}} = \{d, e\}$
- **owns**$^{\mathcal{F}} = \{(b, d), (c, e)\}$;
- **Sue**$^{\mathcal{F}} = a$.

Note that in this interpretation all owners are humans. We can also take **human**$^{\mathcal{F}} = \{a, b\}$ and the sentences from the previous slide are still true in $\mathcal{F}$.

(Note that this is not the case for **human**$^{\mathcal{F}} = \{a\}$.)

Satisfiability

Definition A first-order predicate logic sentence G over S is satisfiable if there exists an S-structure $\mathcal{F}$ such that

$$\mathcal{F} \models G$$

Examples

(a) $\exists x.(P(x) \wedge \neg P(x))$ is not satisfiable.

(b) $\forall x.\exists y.Q(x, y) \wedge \neg\forall u.\exists v.Q(v, u)$ is satisfiable. (The sentence states that the domain of Q is the whole domain of discourse and that the range of Q is not the whole domain of discourse.)

(c) $\forall x.P(x) \wedge \exists x.\neg P(x)$ is not satisfiable.

Proof for (b)

To show that $\forall x.\exists y.Q(x,y) \wedge \neg\forall u.\exists v.Q(v,u)$ is satisfiable we have to define a $\{Q\}$-structure $\mathcal{F}$ such that

$$\mathcal{F} \models \forall x.\exists y.Q(x,y) \wedge \neg\forall u.\exists v.Q(v,u)$$

Let

- $D^{\mathcal{F}} = \{a,b\}$;
- $Q^{\mathcal{F}} = \{(a,b),(b,b)\}$.

We have: for all $d \in D^{\mathcal{F}}$ there exists $d' \in D^{\mathcal{F}}$ with $(d,d') \in Q^{\mathcal{F}}$. Thus, $\mathcal{F} \models \forall x \exists y.Q(x,y)$.

For $d = a$ there does not exists $d' \in D^{\mathcal{F}}$ such that $(d',d) \in Q^{\mathcal{F}}$. Thus, $\mathcal{F} \models \neg\forall u.\exists v.Q(v,u)$.

Proof for (c)

To prove that $\forall x.P(x) \wedge \exists x.\neg P(x)$ is not satisfiable, we have to show that for all $\{P\}$-structures $\mathcal{F}$:

$$\mathcal{F} \not\models \forall x.P(x) \wedge \exists x.\neg P(x)$$

For a proof by contradiction assume there exists a $\{P\}$-structure $\mathcal{F}$ such that

$$\mathcal{F} \models \forall x.P(x) \wedge \exists x.\neg P(x)$$

Since $\mathcal{F} \models \exists x.\neg P(x)$, there exists $d \in D^{\mathcal{F}}$ such that $d \notin P^{\mathcal{F}}$. Take an assignment a with $a(x) = d$. Then $(\mathcal{F}, a) \not\models P(x)$. Hence $\mathcal{F} \not\models \forall x.P(x)$ and we have derived a contradiction to $\mathcal{F} \models \forall x.P(x) \wedge \exists x.\neg P(x)$.

Undecidability of satisfiability (very informal!)

We show that there does not exist any algorithm (no computer program) that decides whether a first-order predicate logic sentence is satisfiable; i.e., for input sentence G outputs "Yes" if G is satisfiable and "No" if G is not satisfiable.

To this end, we show that there is no computer program that decides whether a given computer program P **halts** for a given input number n. (Called **undecidability of the halting problem**.)

To formulate this a bit more precisely, let L be some standard programming language. If a problem is decidable, then one can implement a program in L solving it. With every program P in L we associate a unique number $P^\sharp$.

Undecidability

Now we show: there is no program in L which outputs "yes" for input (P, n) (P a program in L and n a number) if program P terminates for input n, and outputs "no" otherwise.

For a proof by contradiction, assume there exists a program in L which outputs "yes" for input (P, n) if P terminates for input n and outputs "no" otherwise.

Then there exists a program Q_{term} that outputs "yes" for input $P^\sharp$ if P terminates for input $P^\sharp$. Otherwise it outputs "no".

Undecidability

We can rewrite Q_{term} into a program A_{term} that terminates for input $P^{\sharp}$ if Q_{term} outputs "no" for input $P^{\sharp}$. Otherwise it does not terminate. Then

- A_{term} terminates for input $A^{\sharp}_{term}$ if, and only if (by def. of A_{term})
- Q_{term} outputs "no" for input $A^{\sharp}_{term}$ if, and only if (by def. of Q_{term})
- A_{term} does not terminate for input $A^{\sharp}_{term}$.

We have derived a contradiction.

Now, for any (P, n), (P in a standard programming language L), one can write up a first-order predicate logic sentence $F_{P,n}$ such that $F_{P,n}$ is satisfiable if, and only if, P does not terminate for input n. $F_{P,n}$ describes the computation steps of P.

Since the halting problem is undecidable, we obtain that satisfiablility of first-order predicate logic sentences is undecidable.

Tautology

Definition A first-order predicate logic sentence G over S is a tautology if $\mathcal{F} \models G$ holds for every S-structure $\mathcal{F}$.

Examples of tautologies

(a) $\forall x.P(x) \to \exists x.P(x)$;

(b) $\forall x.P(x) \to P(c)$;

(c) $P(c) \to \exists x.P(x)$;

(d) $\forall x(P(x) \leftrightarrow \neg\neg P(x))$;

(e) $\forall x(\neg(P_1(x) \wedge P_2(x)) \leftrightarrow (\neg P_1(x) \vee \neg P_2(x)))$.

Proof for (a)

To show that $\forall x.P(x) \to \exists x.P(x)$ is a tautology it is sufficient to prove that for every $\{P\}$-structure $\mathcal{F}$: if $\mathcal{F} \models \forall x.P(x)$, then $\mathcal{F} \models \exists x.P(x)$.

Assume $\mathcal{F} \models \forall x.P(x)$. Then $P^{\mathcal{F}}$ coincides with the domain of discourse $D^{\mathcal{F}}$; i.e., $P^{\mathcal{F}} = D^{\mathcal{F}}$. Note that $D^{\mathcal{F}}$ is, by definition, always nonempty. Thus, there exists $d \in P^{\mathcal{F}}$. Define a variable assignment a by setting $a(x) = d$. Then $(\mathcal{F}, a) \models P(x)$. Hence $\mathcal{F} \models \exists x.P(x)$.

The relationship between satisfiability and tautologies

Observation

- A first-order predicate logic sentence G is a tautology if, and only if, $\neg G$ is not satisfiable.
- A first-order predicate logic sentence G is satisfiable if, and only if, $\neg G$ is not a tautology.

Consequence There is no algorithm that decides whether a first-order predicate logic sentence is a tautology.

Proof. Such an alogithm could be used to decide satisfiable of first-order predicate logic sentences. As satisfiability of first-order predicate logic sentences is undecidable, being a tautology is undecidable as well.

Semantic consequence

Definition Let X be a set of sentences over a signature S and G be a sentence over S. Then **G follows from X** (is a semantic consequence of X) if the following implication holds for every S-structure $\mathcal{F}$:

$$\text{If } \mathcal{F} \models E \text{ for all } E \in X\text{, then } \mathcal{F} \models G.$$

This is denoted by

$$X \models G$$

Observations

- For any first-order sentence G: $\emptyset \models G$ if, and only if, G is a tautology. Since 'being a tautology' is undecidable it follows that 'being a logical consequence' is undecidable: there is not algorithm that decides whether $X \models G$.
- $\{G\} \models F$ if, and only if, $G \to F$ is a tautology.

Examples

Let c be an individual constant and P, Q unary predicate symbols.

- We have:
$$\{P(c)\} \models \exists x P(x).$$
- We have:
$$\{P(c), \forall x(P(x) \to Q(x))\} \models \exists x Q(x).$$
- We have:
$$\{P(c), \exists x(P(x) \wedge Q(x))\} \not\models Q(c).$$

Example

We can model the logical consequence "if my car is blue and all blue cars are fast, then my car is fast" as follows: take

- individual constant **MyCar** and
- unary predicates **blue**, **fast**, **car**.

Then

$$\{\mathbf{blue}(\mathbf{MyCar}), \mathbf{car}(\mathbf{MyCar}), \forall x.((\mathbf{blue}(x) \wedge \mathbf{car}(x)) \rightarrow \mathbf{fast}(x))\} \models \mathbf{fast}(\mathbf{MyCar})$$

$\mathcal{F} \models G$ versus $X \models G$

- Note that $\mathcal{F} \models G$ or $\mathcal{F} \models \neg G$, for every sentence G. Thus, we have **complete** information about the domain of discourse. There are many examples where $X \not\models G$ and $X \not\models \neg G$. We have **incomplete** information.

- $\mathcal{F} \models G$ means that G is true in the structure $\mathcal{F}$. Checking whether this is the case for finite $\mathcal{F}$ coincides with **querying relational database instances** and can be done very efficiently. It is also the underlying problem of **model checking** approaches to program verification: $\mathcal{F}$ is a representation of a program and one wants to know whether a property expressed by G is true.

- $X \models G$ means that G is true in every structure in which X is true. This is a much harder problem; in fact, it is undecidable. The research area **automated reasoning** develops methods which work in some cases. Incomplete databases, ontologies, and theorem proving are application areas which are based on the problem of deciding $X \models G$.

Logical equivalence

Definition Two first-order predicate logic sentences G_1 and G_2 over a signature S are called **logically equivalent** if they are true in the same S-structures. In other words, G_1 and G_2 are logically equivalent if the following holds for all S-structures $\mathcal{F}$:

$$\mathcal{F} \models G_1 \text{ if and only if } \mathcal{F} \models G_2.$$

This is denoted by

$$G_1 \equiv G_2.$$

Observations

- $G_1 \equiv G_2$ if, and only if, $\{G_1\} \models G_2$ and $\{G_2\} \models G_1$;
- $G_1 \equiv G_2$ if, and only if, $G_1 \leftrightarrow G_2$ is a tautology.

Examples

Let G and H be first-order predicate logic formulas.

- $\neg\forall x.G \equiv \exists x.\neg G$;
- $\neg\exists x.G \equiv \forall x.\neg G$;
- $\exists x.G \equiv \neg\forall x.\neg G$;
- $\forall x.G \equiv \neg\exists x.\neg G$;
- $\forall x.G \wedge \forall x.H \equiv \forall x.(G \wedge H)$;
- $\forall x.G \vee \forall x.H \not\equiv \forall x.(G \vee H)$;
- $\exists x.G \vee \exists x.H \equiv \exists x(G \vee H)$;
- $\exists x.G \wedge \exists x.H \not\equiv \exists x.(G \wedge H)$.

Completeness of First-order Predicate Logic

Theorem There exists a computer program that outputs exactly the tautologies of first-order predicate logic.

There are many distinct types of **deduction systems** that can be used to implement such a program:

- Hilbert-style systems
- Natural deduction
- Sequent calculus
- Tableaux method
- Resolution

Hilbert Style System

We assume that formulas are constructed using $\wedge$, $\neg$, and $\forall$. We regard $\vee$, $\rightarrow$, and $\exists$ as abbreviations:

- $(G \vee D) = \neg(\neg G \wedge \neg D)$;
- $(G \rightarrow D) = (\neg G \vee D)$;
- $\exists x.G = \neg\forall x.\neg G$.

We set

$$(G \rightarrow D \rightarrow F) = (G \rightarrow (D \rightarrow F))$$

Intuitively, $(G \rightarrow D \rightarrow F)$ means "If G and D are true, then F is true".

We slightly generalise the notion of a tautology: a formula G is called a tautology if, and only if, the sentence $\forall x_1. \dots \forall x_n.G$ is a tautology, where $x_1, \dots, x_n$ are the free varables of G.

The Axioms (Part 1)

The following formulas are tautologies, for any first-order predicate logic formulas $\boldsymbol{D}$, $\boldsymbol{G}$, and $\boldsymbol{F}$:

(A1) $((\boldsymbol{G} \to \boldsymbol{D} \to \boldsymbol{F}) \to (\boldsymbol{G} \to \boldsymbol{D}) \to (\boldsymbol{G} \to \boldsymbol{F}))$

(A2) $(\boldsymbol{G} \to (\boldsymbol{D} \to \boldsymbol{G} \wedge \boldsymbol{D}))$

(A3) $((\boldsymbol{G} \wedge \boldsymbol{D}) \to \boldsymbol{G})$

(A4) $((\boldsymbol{G} \wedge \boldsymbol{D}) \to \boldsymbol{D})$

(A5) $((\boldsymbol{G} \to \neg \boldsymbol{D}) \to (\boldsymbol{D} \to \neg \boldsymbol{G}))$

Note that (A1)-(A5) “axiomatize the propositional part” of first-order predicate logic.

The Axioms (Part 2)

If t is a term, then we write $G\frac{t}{x}$ for the formula obtained from G by replacing every free occurrence of x in G by t. For example

- $P(x)\frac{c}{x} = P(c)$
- $\forall x.P(x)\frac{c}{x} = \forall x.P(x)$

The following formulas are tautologies for all first-order predicate logic formulas G and D:

(A6) $\forall x.G \rightarrow G\frac{t}{x}$ (t not bound in G)

(A7) $G \rightarrow \forall x.G$ (x not free in G)

(A8) $(\forall x.(G \rightarrow D) \rightarrow (\forall x.G \rightarrow \forall x.D))$

(A9) $(\forall y.G\frac{y}{x} \rightarrow \forall x.G)$ (y not in G)

Two Rules

We take, in addition to the axioms (A1)-(A9), the following two rules:

(MP) $G, G \to D / D$;

(GE) $G / \forall x.G$.

The first rules is called **Modus Ponens**. The second rule is called **Generalisation**.

Note

- if G and $G \to D$ are tautologies, then D is a tautology
- if G is a tautology, then $\forall x.G$ is a tautology.

Completeness

We say that G is **provable from (A1)-(A9) with (MP) and (GE)** if, and only if, there is a sequence $G_1, \ldots, G_n$ such that for all G_i one of the following conditions holds:

- G_i is of the form (A1) or (A2) or $\cdots$ or (A9);
- there exists $j < i$ such that $G_j = F$ and $G_i = \forall x.F$ (for some F, x);
- there exist $j, k < i$ such that $G_j = F$ and $G_k = F \to G_i$ (for some F).

Theorem A formula G is a tautology if, and only if, it is provable from (A1)-(A9) with (MP) and (GE).

Arithmetic is incomplete

Recall that

$$S_{AR} = \{\textbf{smaller}, \textbf{sum}, \textbf{prod}, \textbf{even}, \underline{0}, \underline{1}, \ldots\}$$

The standard structure for S_{AR} is given by

$$\mathcal{A} = (D^{\mathcal{A}}, \textbf{smaller}^{\mathcal{A}}, \textbf{sum}^{\mathcal{A}}, \textbf{prod}^{\mathcal{A}}, \textbf{even}^{\mathcal{A}}, \underline{0}^{\mathcal{A}}, \underline{1}^{\mathcal{A}}, \ldots)$$

where

- $D^{\mathcal{A}}$ is the set of natural numbers $0, 1, 2 \ldots$;
- $\textbf{smaller}^{\mathcal{A}} = \{(n, m) \mid n < m\}$;
- $\textbf{sum}^{\mathcal{A}} = \{(n, m, k) \mid n + m = k\}$;
- $\textbf{prod}^{\mathcal{A}} = \{(n, m, k) \mid n \times m = k\}$;
- $\textbf{even}^{\mathcal{A}} = \{n \mid n \text{ is even }\}$;
- $\underline{0}^{\mathcal{A}} = 0$, $\underline{1}^{\mathcal{A}} = 1$, and so on.

Incompleteness of Arithmetic

Theorem There does not exist an algorithm (computer program) that outputs exactly the first-order sentences G over S_{AR} such that $\mathcal{A} \models G$.

It follows, in particular, that there is no finitary calculus that captures the sentences that are true in $\mathcal{A}$. Undecidability of arithmetic follows from incompleteness:

Theorem There does not exist an algorithm (computer program) that decides whether for a first-order sentence G over S_{AR} it holds that $\mathcal{A} \models G$.

Second-order Predicate Logic

Second-order predicate logic is an extension of first-order predicate logic which has, for every arity $n > 0$, variables X for relations of arity n. If X is a variable of arity n and $t_1, \ldots, t_n$ are terms, then

- $X(t_1, \ldots, t_n)$ is a second-order formula.

Then, if X is a variable and G is a second-order predicate logic formula,

- $\forall X.G$ is a formula of second-order predicate logic.

Second-order logic is much more expressive than first-order logic. For example, one can formalise induction (we use, for simplicity, $+$ instead of **sum**):

$$\forall X.((X(0) \wedge (\forall x.X(x) \to X(x+1))) \to \forall y.X(y))$$

Theorem There does not exist an algorithm (computer program) that outputs exactly the tautologies of second-order logic.