3 First-Order Logic

First-order logic
• formalizes fundamental mathematical concepts
• is expressive (Turing-complete)
• is not too expressive (e.g. not axiomatizable: natural numbers, uncountable sets)
• has a rich structure of decidable fragments
• has a rich model and proof theory

First-order logic is also called (first-order) predicate logic.

3.1 Syntax

Syntax:
• non-logical symbols (domain-specific) ⇒ terms, atomic formulas
• logical connectives (domain-independent) ⇒ Boolean combinations, quantifiers

Signature

A signature Σ = (Ω, Π) fixes an alphabet of non-logical symbols, where
• Ω is a set of function symbols f with arity n ≥ 0, written arity(f) = n,
• Π is a set of predicate symbols P with arity m ≥ 0, written arity(P) = m.

Function symbols are also called operator symbols.
If n = 0 then f is also called a constant (symbol).
If m = 0 then P is also called a propositional variable.

We will usually use
b, c, d for constant symbols,
f, g, h for non-constant function symbols,
P, Q, R, S for predicate symbols.
Convention: We will usually write \( f/n \in \Omega \) instead of \( f \in \Omega \), \( \text{arity}(f) = n \) (analogously for predicate symbols).

Refined concept for practical applications: *many-sorted* signatures (corresponds to simple type systems in programming languages); not so interesting from a logical point of view.

**Variables**

Predicate logic admits the formulation of abstract, schematic assertions. (Object) variables are the technical tool for schematization.

We assume that \( X \) is a given countably infinite set of symbols which we use to denote variables.

**Terms**

Terms over \( \Sigma \) and \( X \) (\( \Sigma \)-terms) are formed according to these syntactic rules:

\[
s, t, u, v ::= x, \quad x \in X \quad \text{(variable)} \]
\[
\mid f(s_1, \ldots, s_n), \quad f/n \in \Omega \quad \text{(functional term)}
\]

By \( T_\Sigma(X) \) we denote the set of \( \Sigma \)-terms (over \( X \)). A term not containing any variable is called a ground term. By \( T_\Sigma \) we denote the set of \( \Sigma \)-ground terms.

In other words, terms are formal expressions with well-balanced brackets which we may also view as marked, ordered trees. The markings are function symbols or variables. The nodes correspond to the subterms of the term. A node \( v \) that is marked with a function symbol \( f \) of arity \( n \) has exactly \( n \) subtrees representing the \( n \) immediate subterms of \( v \).

**Atoms**

Atoms (also called atomic formulas) over \( \Sigma \) are formed according to this syntax:

\[
A, B ::= P(s_1, \ldots, s_m), \quad P/m \in \Pi \quad \text{(non-equational atom)}
\]
\[
\lbrack \mid (s \approx t), \quad (s \approx t) \quad \text{(equation)} \rbrack
\]

Whenever we admit equations as atomic formulas we are in the realm of first-order logic *with equality*. Admitting equality does not really increase the expressiveness of first-order logic, (cf. exercises). But deductive systems where equality is treated specifically are much more efficient.
Literals

\[ L ::= A \quad \text{(positive literal)} \]
\[ \quad | \quad \neg A \quad \text{(negative literal)} \]

Clauses

\[ C, D ::= \bot \quad \text{(empty clause)} \]
\[ \quad | \quad L_1 \lor \ldots \lor L_k, \quad k \geq 1 \quad \text{(non-empty clause)} \]

General First-Order Formulas

\( F_\Sigma(X) \) is the set of first-order formulas over \( \Sigma \) defined as follows:

\[ F, G, H ::= \bot \quad \text{(falsum)} \]
\[ \quad | \quad \top \quad \text{(verum)} \]
\[ \quad | \quad A \quad \text{(atomic formula)} \]
\[ \quad | \quad \neg F \quad \text{(negation)} \]
\[ \quad | \quad (F \land G) \quad \text{(conjunction)} \]
\[ \quad | \quad (F \lor G) \quad \text{(disjunction)} \]
\[ \quad | \quad (F \rightarrow G) \quad \text{(implication)} \]
\[ \quad | \quad (F \leftrightarrow G) \quad \text{(equivalence)} \]
\[ \quad | \quad \forall x F \quad \text{(universal quantification)} \]
\[ \quad | \quad \exists x F \quad \text{(existential quantification)} \]

Notational Conventions

We omit brackets according to the conventions for propositional logic.

\[ \forall x_1, \ldots, x_n F \quad \text{and} \quad \exists x_1, \ldots, x_n F \] abbreviate \( \forall x_1 \ldots \forall x_n F \) and \( \exists x_1 \ldots \exists x_n F \).

We use infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences.

Examples:
\[ s + t \cdot u \quad \text{for} \quad + (s, \cdot (t, u)) \]
\[ s \cdot u \leq t + v \quad \text{for} \quad \leq (\cdot (s, u), +(t, v)) \]
\[ -s \quad \text{for} \quad -(s) \]
\[ 0 \quad \text{for} \quad 0() \]
Example: Peano Arithmetic

$$\Sigma_{PA} = (\Omega_{PA}, \Pi_{PA})$$
$$\Omega_{PA} = \{0/0, +/2, */2, s/1\}$$
$$\Pi_{PA} = \{\leq/2, </2\}$$

+, *, <, ≤ infix; * >p, +p, <p, ≤p

Examples of formulas over this signature are:

$$\forall x, y (x \leq y \leftrightarrow \exists z (x + z \approx y))$$
$$\exists x \forall y (x + y \approx y)$$
$$\forall x, y (x * s(y) \approx x * y + x)$$
$$\forall x, y (s(x) \approx s(y) \rightarrow x \approx y)$$
$$\forall x \exists y (x < y \land \neg \exists z (x < z \land z < y))$$

Remarks About the Example

We observe that the symbols ≤, <, 0, s are redundant as they can be defined in first-order logic with equality just with the help of +. The first formula defines ≤, while the second defines zero. The last formula, respectively, defines s.

Eliminating the existential quantifiers by Skolemization (cf. below) reintroduces the “redundant” symbols.

Consequently there is a trade-off between the complexity of the quantification structure and the complexity of the signature.

Positions in Terms and Formulas

The set of positions is extended from propositional logic to first-order logic:

The positions of a term s (formula F):

$$\text{pos}(x) = \{\varepsilon\},$$
$$\text{pos}(f(s_1, \ldots, s_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i p \mid p \in \text{pos}(s_i)\},$$
$$\text{pos}(P(t_1, \ldots, t_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i p \mid p \in \text{pos}(t_i)\},$$
$$\text{pos}(\forall x F) = \{\varepsilon\} \cup \{1p \mid p \in \text{pos}(F)\},$$
$$\text{pos}(\exists x F) = \{\varepsilon\} \cup \{1p \mid p \in \text{pos}(F)\}.$$

The prefix order ≤, the subformula (subterm) operator, the formula (term) replacement operator and the size operator are extended accordingly. See the definitions in Sect. 2.
Bound and Free Variables

In $QxF$, $Q \in \{\exists, \forall\}$, we call $F$ the scope of the quantifier $Qx$. An occurrence of a variable $x$ is called bound, if it is inside the scope of a quantifier $Qx$. Any other occurrence of a variable is called free.

Formulas without free variables are also called closed formulas or sentential forms.

Formulas without variables are called ground.

Example:

$$\forall y \ (\forall x \ P(x) \rightarrow Q(x, y))$$

The occurrence of $y$ is bound, as is the first occurrence of $x$. The second occurrence of $x$ is a free occurrence.

Substitutions

Substitution is a fundamental operation on terms and formulas that occurs in all inference systems for first-order logic.

In general, substitutions are mappings

$$\sigma : X \rightarrow T_X(X)$$

such that the domain of $\sigma$, that is, the set

$$\text{dom}(\sigma) = \{ x \in X \mid \sigma(x) \neq x \},$$

is finite. The set of variables introduced by $\sigma$, that is, the set of variables occurring in one of the terms $\sigma(x)$, with $x \in \text{dom}(\sigma)$, is denoted by $\text{codom}(\sigma)$.

Substitutions are often written as $\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$, with $x_i$ pairwise distinct, and then denote the mapping

$$\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}(y) = \begin{cases} s_i, & \text{if } y = x_i \\ y, & \text{otherwise} \end{cases}$$

We also write $x\sigma$ for $\sigma(x)$.

The modification of a substitution $\sigma$ at $x$ is defined as follows:

$$\sigma[x \mapsto t](y) = \begin{cases} t, & \text{if } y = x \\ \sigma(y), & \text{otherwise} \end{cases}$$
Why Substitution is Complicated

We define the application of a substitution $\sigma$ to a term $t$ or formula $F$ by structural induction over the syntactic structure of $t$ or $F$ by the equations depicted on the next page.

In the presence of quantification it is surprisingly complex: We need to make sure that the (free) variables in the codomain of $\sigma$ are not captured upon placing them into the scope of a quantifier $Qy$, hence the bound variable must be renamed into a “fresh”, that is, previously unused, variable $z$.

Application of a Substitution

“Homomorphic” extension of $\sigma$ to terms and formulas:

$$f(s_1, \ldots, s_n)\sigma = f(s_1\sigma, \ldots, s_n\sigma)$$
$$\bot \sigma = \bot$$
$$\top \sigma = \top$$
$$P(s_1, \ldots, s_n)\sigma = P(s_1\sigma, \ldots, s_n\sigma)$$
$$(u \approx v)\sigma = (u\sigma \approx v\sigma)$$
$$\neg F \sigma = \neg(F\sigma)$$
$$(F \rho G)\sigma = (F\sigma \rho G\sigma) ; \text{ for each binary connective } \rho$$
$$(Qx F)\sigma = Qz (F\sigma[x \mapsto z]) ; \text{ with } z \text{ a fresh variable}$$

3.2 Semantics

To give semantics to a logical system means to define a notion of truth for the formulas. The concept of truth that we will now define for first-order logic goes back to Tarski.

As in the propositional case, we use a two-valued logic with truth values “true” and “false” denoted by 1 and 0, respectively.

Algebras

A $\Sigma$-algebra (also called $\Sigma$-interpretation or $\Sigma$-structure) is a triple

$$\mathcal{A} = (U_\mathcal{A}, (f_\mathcal{A} : U^{f}_{\mathcal{A}} \to U^{m}_{\mathcal{A}})_{f/n \in \Omega}, (P_\mathcal{A} \subseteq U^{m}_{\mathcal{A}})_{P/m \in \Pi})$$

where $U_\mathcal{A} \neq \emptyset$ is a set, called the universe of $\mathcal{A}$.

By $\Sigma$-Alg we denote the class of all $\Sigma$-algebras.
Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A (variable) assignment, also called a valuation (over a given Σ-algebra $A$), is a map $\beta : X \rightarrow U_A$.

Variable assignments are the semantic counterparts of substitutions.

Value of a Term in $A$ with Respect to $\beta$

By structural induction we define

$$A(\beta) : T_\Sigma(X) \rightarrow U_A$$

as follows:

$$A(\beta)(x) = \beta(x), \quad x \in X$$
$$A(\beta)(f(s_1, \ldots, s_n)) = f_A(A(\beta)(s_1), \ldots, A(\beta)(s_n)), \quad f/n \in \Omega$$

In the scope of a quantifier we need to evaluate terms with respect to modified assignments. To that end, let $\beta[x \mapsto a] : X \rightarrow U_A$, for $x \in X$ and $a \in U_A$, denote the assignment

$$\beta[x \mapsto a](y) = \begin{cases} a & \text{if } x = y \\ \beta(y) & \text{otherwise} \end{cases}$$

Truth Value of a Formula in $A$ with Respect to $\beta$

$A(\beta) : F_\Sigma(X) \rightarrow \{0, 1\}$ is defined inductively as follows:

$$A(\beta)(\bot) = 0$$
$$A(\beta)(\top) = 1$$
$$A(\beta)(P(s_1, \ldots, s_n)) = \begin{cases} 1 & \text{if } (A(\beta)(s_1), \ldots, A(\beta)(s_n)) \in P_A \\ 0 & \text{else} \end{cases}$$
$$A(\beta)(s \approx t) = \begin{cases} 1 & \text{if } A(\beta)(s) = A(\beta)(t) \\ 0 & \text{else} \end{cases}$$

39
\[ A(\beta)(\neg F) = 1 - A(\beta)(F) \]
\[ A(\beta)(F \land G) = \min(A(\beta)(F), A(\beta)(G)) \]
\[ A(\beta)(F \lor G) = \max(A(\beta)(F), A(\beta)(G)) \]
\[ A(\beta)(F \rightarrow G) = \max(1 - A(\beta)(F), A(\beta)(G)) \]
\[ A(\beta)(F \leftrightarrow G) = \begin{cases} 1 & \text{if } A(\beta)(F) = A(\beta)(G) \\ 0 & \text{otherwise} \end{cases} \]
\[ A(\beta)(\forall x F) = \min_{a \in U_A} \{ A(\beta[x \mapsto a])(F) \} \]
\[ A(\beta)(\exists x F) = \max_{a \in U_A} \{ A(\beta[x \mapsto a])(F) \} \]

**Example**

The “Standard” Interpretation for Peano Arithmetic:

\[ U_N = \{0, 1, 2, \ldots\} \]
\[ 0_N = 0 \]
\[ s_N : n \mapsto n + 1 \]
\[ +_N : (n, m) \mapsto n + m \]
\[ \times_N : (n, m) \mapsto n \times m \]
\[ \leq_N = \{ (n, m) | n \text{ less than or equal to } m \} \]
\[ <_N = \{ (n, m) | n \text{ less than } m \} \]

Note that \( \mathbb{N} \) is just one out of many possible \( \Sigma_{PA} \)-interpretations.

Values over \( \mathbb{N} \) for Sample Terms and Formulas:

Under the assignment \( \beta : x \mapsto 1, y \mapsto 3 \) we obtain

\[ \mathbb{N}(\beta)(s(x) + s(0)) = 3 \]
\[ \mathbb{N}(\beta)(x + y \approx s(y)) = 1 \]
\[ \mathbb{N}(\beta)(\forall x, y(x + y \approx y + x)) = 1 \]
\[ \mathbb{N}(\beta)(\forall z z \leq y) = 0 \]
\[ \mathbb{N}(\beta)(\forall x \exists y x < y) = 1 \]
3.3 Models, Validity, and Satisfiability

$F$ is valid in $\mathcal{A}$ under assignment $\beta$:

$$\mathcal{A}, \beta \models F \iff \mathcal{A}(\beta)(F) = 1$$

$F$ is valid in $\mathcal{A}$ ($\mathcal{A}$ is a model of $F$):

$$\mathcal{A} \models F \iff \mathcal{A}, \beta \models F, \text{ for all } \beta \in X \rightarrow U_{\mathcal{A}}$$

$F$ is valid (or is a tautology):

$$\models F \iff \mathcal{A} \models F, \text{ for all } \mathcal{A} \in \Sigma\text{-Alg}$$

$F$ is called satisfiable iff there exist $\mathcal{A}$ and $\beta$ such that $\mathcal{A}, \beta \models F$. Otherwise $F$ is called unsatisfiable.

Substitution Lemma

The following propositions, to be proved by structural induction, hold for all $\Sigma$-algebras $\mathcal{A}$, assignments $\beta$, and substitutions $\sigma$.

Lemma 3.1 For any $\Sigma$-term $t$

$$\mathcal{A}(\beta)(t\sigma) = \mathcal{A}(\beta \circ \sigma)(t),$$

where $\beta \circ \sigma : X \rightarrow U_{\mathcal{A}}$ is the assignment $\beta \circ \sigma(x) = \mathcal{A}(\beta)(x\sigma)$.

Proposition 3.2 For any $\Sigma$-formula $F$, $\mathcal{A}(\beta)(F\sigma) = \mathcal{A}(\beta \circ \sigma)(F)$.

Corollary 3.3 $\mathcal{A}, \beta \models F\sigma \iff \mathcal{A}, \beta \circ \sigma \models F$

These theorems basically express that the syntactic concept of substitution corresponds to the semantic concept of an assignment.
Entailment and Equivalence

$F$ entails (implies) $G$ (or $G$ is a consequence of $F$), written $F \models G$, if for all $A \in \Sigma$-Alg and $\beta \in X \rightarrow U_A$, whenever $A, \beta \models F$, then $A, \beta \models G$.

$F$ and $G$ are called equivalent, written $F \equiv G$, if for all $A \in \Sigma$-Alg and $\beta \in X \rightarrow U_A$ we have $A, \beta \models F \iff A, \beta \models G$.

**Proposition 3.4** $F$ entails $G$ iff $(F \rightarrow G)$ is valid

**Proposition 3.5** $F$ and $G$ are equivalent iff $(F \leftrightarrow G)$ is valid.

Extension to sets of formulas $N$ in the “natural way”, e.g., $N \models F$:

$\iff$ for all $A \in \Sigma$-Alg and $\beta \in X \rightarrow U_A$: if $A, \beta \models G$, for all $G \in N$, then $A, \beta \models F$.

Validity vs. Unsatisfiability

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

**Proposition 3.6** Let $F$ and $G$ be formulas, let $N$ be a set of formulas. Then

(i) $F$ is valid if and only if $\neg F$ is unsatisfiable.

(ii) $F \models G$ if and only if $F \land \neg G$ is unsatisfiable.

(iii) $N \models G$ if and only if $N \cup \{ \neg G \}$ is unsatisfiable.

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

Theory of an Algebra

Let $A \in \Sigma$-Alg. The (first-order) theory of $A$ is defined as

$$Th(A) = \{ G \in F_\Sigma(X) \mid A \models G \}$$

Problem of axiomatizability:

For which algebras $A$ can one axiomatize $Th(A)$, that is, can one write down a formula $F$ (or a recursively enumerable set $F$ of formulas) such that

$$Th(A) = \{ G \mid F \models G \}?$$

Analogously for sets of algebras.
Two Interesting Theories

Let $\Sigma_{Pres} = (\{0/0, s/1, +/2\}, \emptyset)$ and $\mathbb{Z}_+ = (\mathbb{Z}, 0, s, +)$ its standard interpretation on the integers. $Th(\mathbb{Z}_+)$ is called Presburger arithmetic (M. Presburger, 1929). (There is no essential difference when one, instead of $\mathbb{Z}$, considers the natural numbers $\mathbb{N}$ as standard interpretation.)

Presburger arithmetic is decidable in 3EXPTIME (D. Oppen, JCSS, 16(3):323–332, 1978), and in 2EXPSPACE, using automata-theoretic methods (and there is a constant $c \geq 0$ such that $Th(\mathbb{Z}_+) \notin \text{NTIME}(2^{2^{cn}})$).

However, $\mathbb{N} = (\mathbb{N}, 0, s, +, *)$, the standard interpretation of $\Sigma_{PA} = (\{0/0, s/1, +/2, */2\}, \emptyset)$, has as theory the so-called Peano arithmetic which is undecidable, not even recursively enumerable.

Note: The choice of signature can make a big difference with regard to the computational complexity of theories.
3.4 Algorithmic Problems

Validity($F$): $\models F$?

Satisfiability($F$): $F$ satisfiable?

Entailment($F,G$): does $F$ entail $G$?

Model($A,F$): $A \models F$?

Solve($A,F$): find an assignment $\beta$ such that $A,\beta \models F$.

Solve($F$): find a substitution $\sigma$ such that $\models F\sigma$.

Abduce($F$): find $G$ with “certain properties” such that $G \models F$.

Gödel’s Famous Theorems

1. For most signatures $\Sigma$, validity is undecidable for $\Sigma$-formulas. (One can easily encode Turing machines in most signatures.)

2. For each signature $\Sigma$, the set of valid $\Sigma$-formulas is recursively enumerable. (We will prove this by giving complete deduction systems.)

3. For $\Sigma = \Sigma_{PA}$ and $\mathbb{N}_* = (\mathbb{N}, 0, s, +, \star)$, the theory $Th(\mathbb{N}_*)$ is not recursively enumerable.

These complexity results motivate the study of subclasses of formulas (fragments) of first-order logic.

Q: Can you think of any fragments of first-order logic for which validity is decidable?

Some Decidable Fragments

Some decidable fragments:

- **Monadic class**: no function symbols, all predicates unary; validity is NEXPTIME-complete.

- Variable-free formulas without equality: satisfiability is NP-complete. (why?)

- Variable-free Horn clauses (clauses with at most one positive atom): entailment is decidable in linear time.

- Finite model checking is decidable in time polynomial in the size of the algebra and the formula.