First-Order Logic

Peter A. Flach
Department of Computer Science, University of Bristol, Bristol, UK

Synonyms
Predicate logic; Predicate calculus; First-order predicate logic; First-order predicate calculus

Definition
First-order predicate logic – first-order logic for short – is the logic of properties of, and relations between, objects and their parts. Like any logic, it consists of three parts: syntax governs the formation of well-formed formulae, semantics ascribes meaning to well-formed formulae and formalizes the notion of deductive consequence, and proof procedures allow the inference of deductive consequences by syntactic means. A number of variants of first-order logic exist, mainly differing in their syntax and proof systems. In machine learning, the main use of first-order logic is in Learning from Structured Data, Inductive Logic Programming, and Relational Data Mining.

Motivation and Background
The interest in logic arises from a desire to formalize human, mathematical, and scientific reasoning and goes back to at least the Greek philosophers. Aristotle devised a form of propositional reasoning called syllogisms in the fourth century BC. Aristotle was held in very high esteem by medieval scholars, and so further significant advances were not made until after the Middle Ages. Leibniz wrote of an “algebra of thought” and linked reasoning to calculation in the late seventeenth century. Boole and De Morgan developed the algebraic point of view in the mid-nineteenth century.

Universally quantified variables, which form the main innovation in first-order logic as compared to Propositional Logic, were invented by Gottlob Frege in his Begriffsschrift (“concept notation”) from 1879 and independently by Charles Sanders Peirce in 1885, who introduced the notation \( \Pi_x \) and \( \Sigma_x \) for universal and existential quantification. Frege’s work went largely unnoticed until it was developed further by Alfred North Whitehead and Bertrand Russell in their Principia Mathematica (1903). Seminal contributions were made, among many others: by Giuseppe Peano, who axiomatized number theory and introduced the notation \( (x) \) and \( \exists x \); by Kurt Gödel, who established the completeness of first-order logic as well as the incompleteness
of any system incorporating Peano arithmetic; by Alonzo Church, who proved that first-order logic is undecidable and who introduced λ-calculus, a form of Higher-Order Logic that allows quantification over predicates and functions (as opposed to first-order logic, which only allows quantification over objects); and by Alfred Tarski, who pioneered logical semantics through model theory and the notion of logical consequence. The now universally accepted notation \( \forall x \) was introduced by Gerhard Gentzen.

Logic plays an important role in any approach to symbolic AI that employs a formal language for knowledge representation and inference. A significant, relatively recent development was the introduction of logic programming languages such as Prolog, which turn logical inference into computation. In machine learning, the use of a first-order language is essential in order to handle domains in which objects have inherent structure; the availability of Prolog as a common language and programming platform gave rise to the field of Inductive Logic Programming.

### Theory

#### Syntax

A first-order logical language is built from constant symbols, variable symbols, predicate symbols, and function symbols; the latter two kinds of symbols have an associated arity, which is the number of arguments they take. Terms are either constant symbols, variable symbols, or of the form \( f(t_1, \ldots, t_n) \) where \( f \) is a function symbol with arity \( n \) and \( t_1, \ldots, t_n \) is a sequence of \( n \) terms. Using the logical connectives \( \neg \) (negation), \( \land \) (conjunction), \( \lor \) (disjunction), and \( \to \) (material implication) and the quantifiers \( \forall \) (universal quantifier) and \( \exists \) (existential quantifier), well-formed formulae or wffs are defined recursively as follows: (1) if \( P \) is a predicate symbol with arity \( n \), and \( t_1, \ldots, t_n \) is a sequence of \( n \) terms, then \( P(t_1, \ldots, t_n) \) is a wff, also referred to as an atomic formula or atom; (2) if \( \phi_1 \) and \( \phi_2 \) are wffs, then \( \neg \phi_1 \), \( \phi_1 \land \phi_2 \), \( \phi_1 \lor \phi_2 \), and \( \phi_1 \to \phi_2 \) are wffs; (3) if \( x \) is a variable and \( \phi \) is a wff, then \( \forall x : \phi \) and \( \exists x : \phi \) are wffs; (4) and nothing else is a wff. Brackets are usually dropped as much as it is possible without causing confusion.

**Example 1** Let “man,” “single,” and “partner” be two unary and one binary predicate symbol, respectively, and let “x” and “y” be variable symbols, then the following is a wff \( \phi \) expressing that men who are not single have a partner:

\[
(\forall x : (\text{man}(x) \land \neg \text{single}(x)))
\to (\exists y : \text{partner}(x, y))
\]

Assuming that \( \neg \) binds strongest, then \( \land \), then \( \to \), the brackets can be dropped:

\[
\forall x : \text{man}(x) \land \neg \text{single}(x)
\to \exists y : \text{partner}(x, y)
\]

A propositional language is a special case of a predicate-logical language, built only from predicate symbols with arity 0, referred to as proposition symbols or propositional atoms, and connectives. So, for instance, assuming the proposition symbols “man,” “single,” and “has_partner,” the following is a propositional wff: \( \text{man} \land \neg \text{single} \to \text{has_partner} \). The main difference is that in propositional logic, references to objects cannot be expressed and therefore have to be understood implicitly.

#### Semantics

First-order wffs express statements that can be true or false, and so a first-order semantics consists in constructing a mapping from wffs to truth values, given an interpretation, which is a possible state of affairs in the domain of discourse, mapping constant, predicate, and function symbols to elements, relations, and functions in and over the domain. To deal with variables, a valuation function is employed. Once this mapping is defined, the meaning of a wff consists in the set of interpretations in which the wff maps to true, also called its models. The intuition is that the more “knowledge” a wff contains, the fewer models it has. The key notion of logical consequence is then defined in terms of models:
one wff is a logical consequence of another if the set of models of the first contains the set of models of the second; hence, the second wff contains at least the same, if not more, knowledge than the first.

Formally, a predicate-logical interpretation, or interpretation for short, is a pair $(D, i)$, where $D$ is a nonempty domain of individuals and $i$ is a function assigning to every constant symbol an element of $D$, to every function symbol with arity $n$ a mapping from $D^n$ to $D$, and to every predicate symbol with arity $n$ a subset of $D^n$, called the extension of the predicate. A valuation is a function $v$ assigning to every variable symbol an element of $D$.

Given an interpretation $I = (D, i)$ and a valuation $v$, a mapping $i_v$ from terms to individuals is defined as follows: (1) if $t$ is a constant symbol, $i_v(t) = i(t)$; (2) if $t$ is a variable symbol, $i_v(t) = v(t)$; (3) and if $t$ is a term $f(t_1, \ldots, t_n)$, $i_v(t) = i(\vec{v})$ for terms $\vec{v}$.

The mapping is extended to a valuation from wffs to truth values as follows: (4) if $\phi$ is an atom $P(t_1, \ldots, t_n)$, $i_v(\phi) = i(\vec{v})$; (5) $i_v(\neg \phi) = T$ if $i_v(\phi) = F$ and $F$ otherwise; (6) $i_v(\phi_1 \land \phi_2) = T$ if $i_v(\phi_1) = T$ and $i_v(\phi_2) = T$ and $F$ otherwise; (7) and $i_v(\forall x: \phi) = T$ if $i_v(x) = \phi$ for all $x \in D$ and $F$ otherwise, where $v_{x=d}$ is $v$ except that $x$ is assigned $d$.

The remaining connectives and quantifiers are evaluated by rewriting: (8) $i_v(\phi_1 \lor \phi_2) = i_v(\neg(\neg \phi_1 \land \neg \phi_2))$; (9) $i_v(\phi_1 \to \phi_2) = i_v(\neg \phi_1 \lor \phi_2)$; (10) $i_v(\exists x: \phi) = i_v(\neg \forall x: \neg \phi)$.

An interpretation $I$ satisfies a wff $\phi$, notation $I \models \phi$, if $i_v(\phi) = T$ for all valuations $v$; we say that $I$ is a model of $\phi$ and that $\phi$ is satisfiable. If all models of a set of wffs $\Sigma$ are also models of $\phi$, we say that $\Sigma$ logically entails $\phi$ or $\phi$ is a logical consequence of $\Sigma$ and write $\Sigma \models \phi$.

If $\Sigma = \emptyset$, $\phi$ is called a tautology and we write $\models \phi$. A wff $\psi$ is a contradiction if $\neg \psi$ is a tautology. Contradictions do not have any models, and consequently $\models \phi$ for any wff $\phi$.

The deduction theorem says that $\Sigma \models \phi$ if and only if $\Sigma \cup \{\phi\} \models \beta$. Another useful fact is that, if $\Sigma \cup \{\neg \gamma\}$ is a contradiction, $\Sigma \models \gamma$; this gives rise to a proof technique known as reductio ad absurdum or proof by contradiction (see below).

Example 2 We continue the previous example. Let $D = \{Peter, Paul, Mary\}$, and let the function $i$ be defined as follows: $i(man) = \{Peter, Paul\}$; $i(single) = \{Paul\}$; $i(partner) = \{(Peter, Mary)\}$. We then have that the interpretation $I = (D, i)$ is a model for the wff $\phi$ above. On the other hand, $I$ does not satisfy $\psi = \forall x : \exists y : partner(x, y)$, and therefore $\phi \not\models \psi$. However, the reverse does hold: there is no interpretation that satisfies $\psi$ and not $\phi$, and therefore $\psi \models \phi$.

In case of a propositional logic, this semantics can be considerably simplified. Since there are no terms, the domain $D$ plays no role, and an interpretation simply assigns truth values to proposition symbols. Wffs can then be evaluated using rules (5–6) and (8–9). For example, if $i(man) = T$, $i(single) = T$, and $i(has\_partner) = T$, then $i(man \land single \to has\_partner) = T$. (If this seems counterintuitive, this is probably because the reader’s knowledge of the domain suggests another wff $\neg(single \land has\_partner)$, which is false in this particular interpretation.)

Proofs

A proof procedure consists of a set of axioms and a set of inference rules. Given a proof procedure $P$, we say that $\phi$ is provable from $\Sigma$ and write $\Sigma \vdash_P \phi$ if there exists a finite sequence of wffs $\phi_1, \phi_2, \ldots, \phi_{n-1}, \phi$ which is obtained by successive applications of inference rules to axioms, premises in $\Sigma$, and/or previous wffs in the sequence. Such a sequence of wffs, if it exists, is called a proof of $\phi$ from $\Sigma$. A proof procedure $P$ is sound, with respect to the semantics established by predicate-logical interpretations, if $\Sigma \models \phi$ whenever $\Sigma \vdash_P \phi$; it is complete if $\Sigma \vdash_P \phi$ whenever $\Sigma \models \phi$. For a sound and complete proof procedure for first-order predicate logic, see, e.g., Turner (1984, p.15).

A set of wffs $\Sigma$ is consistent, with respect to a proof procedure $P$, if not both $\Sigma \vdash_P \phi$ and $\Sigma \vdash_P \neg \phi$ for some wff $\phi$. Given a sound and complete proof procedure, the proof-theoretic notion of consistency coincides with the semantic notion

\textit{First-Order Logic 3}
of satisfiability. In particular, if we can prove that $\Sigma \cup \{\neg \gamma\}$ is inconsistent, then we know that $\Sigma \cup \{\neg \gamma\}$ is not satisfiable, hence a contradiction, and thus $\Sigma \models \gamma$. This still holds if the proof procedure is only complete in the weaker sense of being able to demonstrate the inconsistency of arbitrary sets of wffs (see the resolution inference rule, below).

**Example 3** One useful inference rule for predicate logic replaces a universally quantified variable with an arbitrary term, which is called **Universal Elimination**. So, if “$c$” is a constant symbol in our language, then we can infer

$$\text{man}(c) \land \neg \text{single}(c) \rightarrow \exists y : \text{partner}(c,y)$$

from $\phi$ above by Universal Elimination. Another inference rule, which was called **Modus Ponens** by Aristotle, allows us to infer $\beta$ from $\alpha$ and $\alpha \rightarrow \beta$. So, if we additionally have $\text{man}(c) \land \neg \text{single}(c)$, then we can conclude

$$\exists y : \text{partner}(c,y)$$

by Modus Ponens. This rule is also applicable to propositional logic. An example of an axiom is $c = c$ for any constant symbol $c$. (Strictly speaking this is an axiom schema, giving rise to an axiom for every constant symbol in the language.)

**Programming in Logic**

Syntax, semantics, and proof procedures for first-order logic can be simplified and made more amenable to computation if we limit the number of ways of expressing the same thing. This can be achieved by restricting wffs to a normal form called **prenex conjunctive normal form** (PCNF). This means that all quantifiers occur at the start of the wff and are followed by a conjunction of disjunctions of atoms and negated atoms, jointly called **literals**. An example of a formula in PCNF is

$$\forall x : \exists y : \neg \text{man}(x) \lor \text{single}(x) \lor \text{partner}(x,y)$$

This formula is equivalent to the wff $\phi$ in Example 1, in the sense that it has the same set of models, and so either one logically entails the other. Every first-order wff can be transformed into a logically equivalent formula in PCNF, which is unique up to the order of conjuncts and disjuncts. A transformation procedure can be found in Flach (1994).

PCNF can be further simplified if we use function symbols instead of existential quantifiers. For instance, instead of $\exists y : \text{partner}(x,y)$, we can say $\text{partner}(x,\text{partner}_of(x))$, where $\text{partner}_of$ is a unary function symbol called a Skolem function, after the Norwegian logician Thoralf Skolem. The two statements are not logically equivalent, as the second entails the first but not vice versa, but this difference is of little practical consequence. Since all variables are now universally quantified, the quantifiers are usually omitted, leading to **clausal form**:

$$\neg \text{man}(x) \lor \text{single}(x) \lor \text{partner}(x,\text{partner}_of(x))$$

To sum up, a wff in clausal form is a conjunction of disjunctions of literals, of which the variables are implicitly universally quantified. The individual disjunctions are called **clauses**.

Further simplifications include dispensing with equality, which means that terms involving function symbols, such as $\text{partner}_of(c)$, are not evaluated and in effect treated as names of objects (in this case, the function symbols are called **functors** or data constructors). Under this assumption each ground term (a term without variables) denotes a different object, which means that we can take the set of ground terms as the domain $D$ of an interpretation; this is called a **Herbrand interpretation**, after the French logician Jacques Herbrand.

The main advantage of clausal logic is the existence of a proof procedure consisting of a single inference rule and no axioms. This inference rule, which is called **resolution**, was introduced by Alan Robinson in 1965 (Robinson 1965). In propositional logic, given two clauses $P \lor Q$ and $\neg Q \lor R$ containing complementary literals
Q and \neg Q, resolution infers the resolvent P \lor R (P and/or R may themselves contain several disjuncts). For instance, given \(\text{man} \lor \text{single} \lor \text{has_partner}\) and \text{man} \lor \text{woman}, we can infer \(\text{woman} \lor \text{single} \lor \text{has_partner}\) by resolution. In first-order logic, Q and \(Q'\) are complementary if Q and \(Q'\) are unifiable, i.e., there exists a substitution \(\theta\) of terms for variables such that \(Q\theta = Q'\theta\), where \(Q\theta\) denotes the application of substitution \(\theta\) to \(Q\); in this case, the resolvent of \(P \lor Q\) and \(\neg Q' \lor R\) is \(P\theta \lor R\theta\). For instance, from the following two clauses:

\[-\text{man}(x) \lor \text{single}(x)
\lor \text{partner}(x, \text{partner_of}(x))
\]

\[-\text{single}(\text{father_of}(c))\]

we can infer

\[-\text{man}(\text{father_of}(c)) \lor \text{partner}(\text{father_of}(c)).
\]

\[\text{partner_of}(\text{father_of}(c))\]

The resolution inference rule is sound but not complete: for instance, it is unable to produce tautologies such as \(\text{man}(c) \lor \neg\text{man}(c)\) if no clauses involving the predicate \text{man} are given. However, it is refutation-complete, which means it can demonstrate the unsatisfiability of any set of clauses by deriving the empty clause, indicated by \(\Box\). For instance, \(\text{man}(c) \land \neg\text{man}(c)\) is a wff consisting of two clauses which are complementary literals, so by resolution we infer the empty clause in one step.

Refutation by resolution is the way in which queries are answered in the logic programming language Prolog. Prolog works with a subset of clausal logic called Horn logic, named after the logician Alfred Horn. A Horn clause is a disjunction of literals with at most one positive (un-negated) literal; Horn clauses can be further divided into definite clauses, which have one positive literal, and goal clauses which have none. A Prolog program consists of definite clauses, and a goal clause functions as a procedure call. Notice that resolving a goal clause with a definite clause results in another goal clause, because the positive literal in the definite clause (also called its head) must be one of the complementary literals. The idea is that the resolution step reformulates the original goal into a new goal that is one step closer to the solution. A refutation is then a sequence of goals \(G, G_1, G_2, \ldots, G_n\) such that \(G\) is the original goal, each \(G_i\) is obtained by resolving \(G_{i-1}\) with a clause from the program \(P\), and \(G_n = \Box\). Such a refutation demonstrates that \(P \cup \{G\}\) is inconsistent, and therefore \(P \models \neg G\).

Finding a refutation amounts to a search problem, because there are typically several program clauses that could be resolved against the current goal. Virtually all Prolog interpreters apply a depth-first search procedure, searching the goal literals left to right and the program clauses top-down. Once a refutation is found, the substitutions collected in all resolution steps are composed to obtain an answer substitution. One unique feature of logic programming is that a goal may have more than one (or, indeed, less than one) refutation and answer substitution from a given program.

**Example 4** Consider the following Prolog program:

```
peano_sum(0, Y, Y).
peano_sum(s(X), Y, s(Z)) :-
    -peano_sum(X, Y, Z).
```

This program defines addition in Peano arithmetic. We follow Prolog syntax: variables start with an uppercase letter, and \(-\) stands for reversed implication \(\leftarrow\) or “if.” The unary functor \(s\) represents the successor function. So the first rule reads “the sum of 0 and an arbitrary number \(y\) is \(y\),” and the second rule reads “the sum of \(x + 1\) and \(y\) is \(z + 1\) if the sum of \(x\) and \(y\) is \(z\).”

The goal \(-\text{peano_sum}(s(0), s(s(0)), Q)\) states, “there are no numbers \(q\) such that \(1 + 2 = q\).” We first resolve this goal with the second program clause to obtain \(-\text{peano_sum}(0, s(s(0)), Z)\) under the substitution \(\{Q / s(Z)\}\). This new goal states, “there are no numbers \(z\) such that \(0 + 2 = z\).” It is resolved with the first clause to yield the empty clause under the substitution \(\{Y / s(s(0)), Z /\)
The resulting answer substitution is \( \{ Q / s(s(s(0))) \} \), i.e., \( q = 3 \).

As another example, goal \(-\text{peano} \_\text{sum}(A,B,s(s(0)))\) states “there are no numbers \( a \) and \( b \) such that \( a + b = 2 \)” This goal has three refutations: one involving the first clause only, yielding the answer substitution \( \{ A / 0, B / s(s(0)) \} \); one involving the second clause then the first, resulting in \( \{ A / s(0), B / s(0) \} \); and the third applying the second clause twice followed by the first, yielding \( \{ A / s(s(0)), B / 0 \} \). Prolog will return these three answers in this order.

Induction in first-order logic amount to reconstructing a logical theory from some of its logical consequences. For techniques to induce a Prolog program given examples such as \( \text{peano} \_\text{sum}(s(0),s(0),s(s(0))) \), see Inductive Logic Programming.

**Cross-References**

- Abduction
- Entailment
- Higher-Order Logic
- Hypothesis Language
- Inductive Logic Programming
- Learning from Structured Data
- Propositionalisation
- Relational Data Mining
- Resolution

**Recommended Reading**

For general introductions to logic and its use in Artificial Intelligence, see Turner (1984) and Genesereth and Nilsson (1987). Kowalski’s classic text *Logic for problem solving* focusses on clausal logic and resolution theorem proving (Kowalski 1979). For introductions to Prolog programming, see Flach (1994) and Bratko (2001).


