Part 2: 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.

2.1 Syntax

Syntax:

- non-logical symbols (domain-specific)
  $\Rightarrow$ terms, atomic formulas
- logical symbols (domain-independent)
  $\Rightarrow$ Boolean combinations, quantifiers

Signature

A signature

$$\Sigma = (\Omega, \Pi),$$

fixes an alphabet of non-logical symbols, where

- Ω is a set of function symbols f with arity $n \geq 0$, written f/n,
- Π is a set of predicate symbols p with arity $m \geq 0$, written p/m.

If $n = 0$ then f is also called a constant (symbol).
If $m = 0$ then p is also called a propositional variable.
We use letters P, Q, R, S, to denote propositional variables.

Signature

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 for (the denotation of) variables.

Terms

Terms over Σ (resp., Σ-terms) are formed according to these syntactic rules:

$$
\begin{array}{rclll}
s, t, u, v & ::= & x & , x \in X & \text{(variable)} \\
& \mid & f(s_1, \ldots, s_n) & , f/n \in \Omega & \text{(functional term)}
\end{array}
$$

By $\mathrm{T}_\Sigma(X)$ we denote the set of Σ-terms (over X).

A term not containing any variable is called a ground term.

By T_Σ we denote the set of Σ-ground terms.

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 Σ are formed according to this syntax:

$$
\begin{array}{rcll}
A, B & ::= & p(s_1, \ldots, s_m) & , \; p/m \in \Pi \\
\Big[& \mid & (s \approx t) & \text{(equation)} \quad \Big]
\end{array}
$$

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 can be much more efficient.

Literals

$$
\begin{array}{lcll}
L & ::= & A & \text{(positive literal)} \\
 & \mid & \neg A & \text{(negative literal)}
\end{array}
$$

Clauses

$$
\begin{array}{llll}
C, D & ::= & \bot & \text{(empty clause)} \\
 & \mid & L_1 \vee \ldots \vee L_k, \;\; k \geq 1 & \text{(non-empty clause)}
\end{array}
$$

General First-Order Formulas

$\mathsf{F}_\Sigma(X)$ is the set of first-order formulas over Σ defined as follows:

$$
\begin{array}{rcll}
F, G, H & ::= & \bot & \text{(falsum)} \\
& \mid & \top & \text{(verum)} \\
& \mid & A & \text{(atomic formula)} \\
& \mid & \neg F & \text{(negation)} \\
& \mid & (F \wedge G) & \text{(conjunction)} \\
& \mid & (F \vee G) & \text{(disjunction)} \\
& \mid & (F \rightarrow G) & \text{(implication)} \\
& \mid & (F \leftrightarrow G) & \text{(equivalence)} \\
& \mid & \forall x F & \text{(universal quantification)} \\
& \mid & \exists x F & \text{(existential quantification)}
\end{array}
$$

Notational Conventions

We omit brackets according to the following rules:

- $\neg \quad >_p \quad \vee \quad >_p \quad \wedge \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$
  (binding precedences)

- $\vee$ and $\wedge$ are associative and commutative

- $\rightarrow$ is right-associative

$Qx_1, \ldots, x_n\, F$ abbreviates $Qx_1 \ldots Qx_n\, F$.

Notational Conventions

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

Examples:

$$
\begin{array}{ccc}
s + t * u & \text{for} & +(s, *(t, u)) \\
s * u \leq t + v & \text{for} & \leq (*(s, u), +(t, v)) \\
-s & \text{for} & -(s) \\
0 & \text{for} & 0()
\end{array}
$$

Example: Peano Arithmetic

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

Examples of formulas over this signature are:

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

Remarks About the Example

We observe that the symbols $\leq$, $<$, 0, s are redundant as they can be defined in first-order logic with equality just with the help of $+$. The first formula defines $\leq$, 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.

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.

Bound and Free Variables

Example:

$$\forall y \quad (\forall \overbrace{x \quad p(x)}^{scope} \quad \rightarrow \quad q(x, y))$$

with the outer scope brace spanning $(\forall x \; p(x) \rightarrow q(x, y))$, labelled *scope*.

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

$$\sigma : X \to \mathsf{T}_\Sigma(X)$$

such that the domain of σ, that is, the set

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

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

Substitutions

Substitutions are often written as $[s_1/x_1, \dots, s_n/x_n]$, with x_i pairwise distinct, and then denote the mapping

$$[s_1/x_1, \dots, s_n/x_n](y) = \begin{cases} s_i, & \text{if } y = x_i \\ y, & \text{otherwise} \end{cases}$$

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

The modification of a substitution σ 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 σ 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 σ 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.

Why this definition of substitution is well-defined will be discussed below.

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

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

Structural Induction

Proposition 2.1:
Let $G = (N, T, P, S)$ be a context-free grammar (possibly infinite) and let q be a property of T^* (the words over the alphabet T of terminal symbols of G).

q holds for *all* words $w \in L(G)$, whenever one can prove the following two properties:

Structural Induction

1. (*base cases*)
   $q(w')$ holds for each $w' \in T^*$ such that $X ::= w'$ is a rule in P.

2. (*step cases*)
   If $X ::= w_0 X_0 w_1 \ldots w_n X_n w_{n+1}$ is in P with $X_i \in N$, $w_i \in T^*$, $n \geq 0$, then for all $w_i' \in L(G, X_i)$, whenever $q(w_i')$ holds for $0 \leq i \leq n$, then also $q(w_0 w_0' w_1 \ldots w_n w_n' w_{n+1})$ holds.

Here $L(G, X_i) \subseteq T^*$ denotes the language generated by the grammar G from the nonterminal X_i.

Structural Recursion

Proposition 2.2:
Let $G = (N, T, P, S)$ be a *unambiguous* (why?) context-free grammar. A function f is well-defined on $L(G)$ (that is, unambiguously defined) whenever these 2 properties are satisfied:

1. (base cases)
   f is well-defined on the words $w' \in \Sigma^*$ for each rule $X ::= w'$ in P.

2. (step cases)
   If $X ::= w_0 X_0 w_1 \ldots w_n X_n w_{n+1}$ is a rule in P then $f(w_0 w_0' w_1 \ldots w_n w_n' w_{n+1})$ is well-defined, assuming that each of the $f(w_i')$ is well-defined.

Substitution Revisited

Q: Does Proposition 2.2 justify that our homomorphic extension

$$\mathit{apply} : \mathsf{F}_\Sigma(X) \times (X \to \mathsf{T}_\Sigma(X)) \quad \to \quad \mathsf{F}_\Sigma(X),$$

with $\mathit{apply}(F, \sigma)$ denoted by $F\sigma$, of a substitution is well-defined?

A: We have two problems here. One is that *"fresh"* is (deliberately) left unspecified. That can be easily fixed by adding an extra variable counter argument to the apply function.

Substitution Revisited

The second problem is that Proposition 2.2 applies to unary functions only. The standard solution to this problem is to curryfy, that is, to consider the binary function as a unary function producing a unary (residual) function as a result:

$$\mathit{apply} : \mathsf{F}_\Sigma(X) \quad \rightarrow \quad ((X \rightarrow \mathsf{T}_\Sigma(X)) \rightarrow \mathsf{F}_\Sigma(X))$$

where we have denoted $(\mathit{apply}(F))(\sigma)$ as $F\sigma$.

E: Convince yourself that this does the trick.

2.2 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.

Structures

A Σ-algebra (also called Σ-interpretation or Σ-structure) is a triple

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

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

Normally, by abuse of notation, we will have $\mathcal{A}$ denote both the algebra and its universe.

By Σ-Alg we denote the class of all Σ-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 $\mathcal{A}$), is a map $\beta : X \to \mathcal{A}$.

Variable assignments are the semantic counterparts of substitutions.

Value of a Term in $\mathcal{A}$ with Respect to β

By structural induction we define

$$\mathcal{A}(\beta) : \mathsf{T}_\Sigma(X) \to \mathcal{A}$$

as follows:

$$\begin{aligned} \mathcal{A}(\beta)(x) &= \beta(x), \qquad x \in X \\ \mathcal{A}(\beta)(f(s_1, \ldots, s_n)) &= f_{\mathcal{A}}(\mathcal{A}(\beta)(s_1), \ldots, \mathcal{A}(\beta)(s_n)), \qquad f/n \in \Omega \end{aligned}$$

Value of a Term in $\mathcal{A}$ with Respect to β

In the scope of a quantifier we need to evaluate terms with respect to modified assignments. To that end, let $\beta[x \mapsto a] : X \to \mathcal{A}$, for $x \in X$ and $a \in \mathcal{A}$, 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 $\mathcal{A}$ with Respect to β

$\mathcal{A}(\beta) : \mathsf{F}_\Sigma(X) \to \{0, 1\}$ is defined inductively as follows:

$$\begin{aligned}
\mathcal{A}(\beta)(\bot) &= 0 \\
\mathcal{A}(\beta)(\top) &= 1 \\
\mathcal{A}(\beta)(p(s_1, \ldots, s_n)) = 1 &\quad\Leftrightarrow\quad (\mathcal{A}(\beta)(s_1), \ldots, \mathcal{A}(\beta)(s_n)) \in p_\mathcal{A} \\
\mathcal{A}(\beta)(s \approx t) = 1 &\quad\Leftrightarrow\quad \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t) \\
\mathcal{A}(\beta)(\neg F) = 1 &\quad\Leftrightarrow\quad \mathcal{A}(\beta)(F) = 0 \\
\mathcal{A}(\beta)(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
&\quad \text{with } \mathsf{B}_\rho \text{ the Boolean function associated with } \rho \\
\mathcal{A}(\beta)(\forall x F) &= \min_{a \in U} \{\mathcal{A}(\beta[x \mapsto a])(F)\} \\
\mathcal{A}(\beta)(\exists x F) &= \max_{a \in U} \{\mathcal{A}(\beta[x \mapsto a])(F)\}
\end{aligned}$$

Example

The "Standard" Interpretation for Peano Arithmetic:

$$\begin{array}{rcl} U_{\mathbb{N}} & = & \{0, 1, 2, \ldots\} \\ 0_{\mathbb{N}} & = & 0 \\ s_{\mathbb{N}} & : & n \mapsto n + 1 \\ +_{\mathbb{N}} & : & (n, m) \mapsto n + m \\ *_{\mathbb{N}} & : & (n, m) \mapsto n * m \\ \leq_{\mathbb{N}} & = & \{(n, m) \mid n \text{ less than or equal to } m\} \\ <_{\mathbb{N}} & = & \{(n, m) \mid n \text{ less than } m\} \end{array}$$

Note that $\mathbb{N}$ is just one out of many possible Σ_{PA}-interpretations.

Example

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

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

$$\begin{array}{lcl}
\mathbb{N}(\beta)(s(x) + s(0)) & = & 3 \\
\mathbb{N}(\beta)(x + y \approx s(y)) & = & 1 \\
\mathbb{N}(\beta)(\forall x, y (x + y \approx y + x)) & = & 1 \\
\mathbb{N}(\beta)(\forall z \; z \leq y) & = & 0 \\
\mathbb{N}(\beta)(\forall x \exists y \; x < y) & = & 1
\end{array}$$

2.3 Models, Validity, and Satisfiability

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

$$\mathcal{A}, \beta \models F \quad :\Leftrightarrow \quad \mathcal{A}(\beta)(F) = 1$$

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

$$\mathcal{A} \models F \quad :\Leftrightarrow \quad \mathcal{A}, \beta \models F, \text{ for all } \beta \in X \to U_{\mathcal{A}}$$

F is valid (or is a tautology):

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

F is called satisfiable iff there exist $\mathcal{A}$ and β 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 Σ-algebras $\mathcal{A}$, assignments β, and substitutions σ.

Lemma 2.3:
For any Σ-term t

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

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

Proposition 2.4:
For any Σ-formula F, $\mathcal{A}(\beta)(F\sigma) = \mathcal{A}(\beta \circ \sigma)(F)$.

Substitution Lemma

Corollary 2.5:

$\mathcal{A}, \beta \models F\sigma \quad \Leftrightarrow \quad \mathcal{A}, \beta \circ \sigma \models F$

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 $F \models G$

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma\text{-Alg}$ and $\beta \in X \to U_{\mathcal{A}}$,
whenever $\mathcal{A}, \beta \models F$ then $\mathcal{A}, \beta \models G$.

F and G are called equivalent

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma\text{-Alg}$ und $\beta \in X \to U_{\mathcal{A}}$ we have
$\mathcal{A}, \beta \models F \quad \Leftrightarrow \quad \mathcal{A}, \beta \models G$.

Entailment and Equivalence

Proposition 2.6:
F entails G iff $(F \rightarrow G)$ is valid

Proposition 2.7:
F and G are equivalent iff $(F \leftrightarrow G)$ is valid.

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

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma$-Alg and $\beta \in X \rightarrow U_{\mathcal{A}}$:
if $\mathcal{A}, \beta \models G$, for all $G \in N$, then $\mathcal{A}, \beta \models F$.

Validity vs. Unsatisfiability

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

Proposition 2.8:

$$F \text{ valid} \quad \Leftrightarrow \quad \neg F \text{ unsatisfiable}$$

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

Q: In a similar way, entailment $N \models F$ can be reduced to unsatisfiability. How?

Theory of a Structure

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

$$Th(\mathcal{A}) = \{G \in \mathsf{F}_\Sigma(X) \mid \mathcal{A} \models G\}$$

Problem of axiomatizability:

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

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

Analogously for sets of structures.

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 $Th(\mathbb{Z}_+) \notin \mathrm{NTIME}(2^{2^{cn}})$).

Two Interesting Theories

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.