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)
  $\Rightarrow$ 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 $\mathsf{arity}(f) = n$,
- Π is a set of *predicate symbols* P with arity $m \geq 0$, written $\mathsf{arity}(P) = m$.

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 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 for (the denotation of) *variables*.

Context-Free Grammars

We define many of our notions on the bases of context-free grammars. Recall, that a context-free grammar $G = (N, T, P, S)$ consists of:

- a set of non-terminal symbols N
- a set of terminal symbols T
- a set P of rules $A ::= w$ where $A \in N$ and $w \in (N \cup T)^*$
- a start symbol S where $S \in N$

For rules $A ::= w_1$, $A ::= w_2$ we write $A ::= w_1 \mid w_2$

Terms

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

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

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.

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:

$$\begin{array}{rcll} A, B & ::= & P(s_1, ..., s_m) & , P \in \Pi, \mathsf{arity}(P) = m \\ \Big[& \mid & (s \approx t) & \text{(equation)} \quad \Big] \end{array}$$

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 are much more efficient.

Literals

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

Clauses

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

General First-Order Formulas

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

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

Positions in terms, formulas

Positions of a term s (formula F):

$\mathrm{pos}(x) = \{\varepsilon\}$,
$\mathrm{pos}(f(s_1,\dots,s_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{\, ip \mid p \in \mathrm{pos}(s_i)\,\}$.

$\mathrm{pos}(\forall x F) = \{\varepsilon\} \cup \{\, 1p \mid p \in \mathrm{pos}(F)\,\}$
Analogously for all other formulas.

Prefix order for $p, q \in \mathrm{pos}(s)$:

p above q: $p \le q$ if $pp' = q$ for some p',
p strictly above q: $p < q$ if $p \le q$ and not $q \le p$,
p and q parallel: $p \parallel q$ if neither $p \le q$ nor $q \le p$.

Subterm of s (F) at a position $p \in \mathrm{pos}(s)$:

$s/\varepsilon = s$,
$f(s_1,\dots,s_n)/ip = s_i/p$.

Analougously for formulas (F/p).

Replacement of the subterm at position $p \in \mathrm{pos}(s)$ by t:

$s[t]_\varepsilon = t$,
$f(s_1,\dots,s_n)[t]_{ip} = f(s_1,\dots,s_i[t]_p,\dots,s_n)$.

Analougously for formulas ($F[G]_p$).

Size of a term s:

$|s| =$ cardinality of $\mathrm{pos}(s)$.

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

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

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

Examples:

$s + t * u$	for	$+(s, *(t, u))$
$s * u \le t + v$	for	$\le (*(s, u), +(t, v))$
$-s$	for	$-(s)$
0	for	$0()$

Example: Peano Arithmetic

$$\begin{aligned} \Sigma_{PA} &= (\Omega_{PA},\ \Pi_{PA}) \\ \Omega_{PA} &= \{0/0,\ +/2,\ */2,\ s/1\} \\ \Pi_{PA} &= \{\le /2,\ < /2\} \end{aligned}$$

$+$, $*$, $<$, $\le$ infix; $* \ >_p \ + \ >_p \ < \ >_p \ \le$

Examples of formulas over this signature are:

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

Remarks About the Example

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

Example:

$$\forall y \quad \overbrace{(\forall \overbrace{x \quad P(x)}^{\textit{scope}} \quad \rightarrow \quad Q(x, y))}^{\textit{scope}}$$

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

$$\sigma : X \rightarrow \mathrm{T}_\Sigma(X)$$

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

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

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

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

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

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:

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

Structural Induction

Proposition 3.1 *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:*

1. *(*base cases*)*
   $q(w')$ holds for each $w' \in T^*$ such that $X ::= w'$ is a rule in P.
2. *(*step cases*)*
   If $X ::= w_0X_0w_1 \dots w_nX_nw_{n+1}$ 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 $q(w_0w'_0w_1 \dots w_nw'_nw_{n+1})$ holds.

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

Structural Recursion

Proposition 3.2 *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 $w' \in T^*$ for each rule $X ::= w'$ in P.
2. *(step cases)*
   If $X ::= w_0 X_0 w_1 \dots w_n X_n w_{n+1}$ is a rule in P then $f(w_0 w'_0 w_1 \dots w_n w'_n w_{n+1})$ is well-defined, assuming that each of the $f(w'_i)$ is well-defined.

Substitution Revisited

Q: Does Proposition 3.2 justify that our homomorphic extension

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

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.

The second problem is that Proposition 3.2 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:

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

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

E: Convince yourself that this does the trick.

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.