

Decision Procedures for Verification

Part 2. First-Order Logic (1)

12.11.2018

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

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)
⇒ terms, atomic formulas
- logical symbols (domain-independent)
⇒ 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).

Most results established for one-sorted signatures extend in a natural way to many-sorted signatures.

Many-sorted Signature

A many-sorted signature

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

fixes an alphabet of non-logical symbols, where

- S is a set of sorts,
- Ω is a set of **function symbols** f with **arity** $a(f) = s_1 \dots s_n \rightarrow s$,
- Π is a set of **predicate symbols** p with **arity** $a(p) = s_1 \dots s_m$

where s_1, \dots, s_n, s_m, s are sorts.

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**.

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**.

Many-sorted case:

We assume that for every sort $s \in S$, X_s is a given countably infinite set of symbols which we use for (the denotation of) **variables** of sort s .

Terms

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

$$\begin{array}{l} t, u, v ::= x, x \in X \quad (\text{variable}) \\ \quad \quad | f(s_1, \dots, s_n), f/n \in \Omega \quad (\text{functional term}) \end{array}$$

By $T_\Sigma(X)$ 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

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

$$\begin{array}{l} t, u, v ::= x, x \in X \quad \text{(variable)} \\ \quad \quad | f(t_1, \dots, t_n), f/n \in \Omega \quad \text{(functional term)} \end{array}$$

By $T_\Sigma(X)$ 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.

Many-sorted case:

a variable $x \in X_s$ is a term of sort s

if $a(f) = s_1 \dots s_n \rightarrow s$, and t_i are terms of sort s_i , $i = 1, \dots, n$ then $f(t_1, \dots, t_n)$ is a term of sort s .

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:

$$A, B ::= p(t_1, \dots, t_m) \quad , p/m \in \Pi \\ \left[\quad \mid \quad (t \approx t') \quad \text{(equation)} \quad \right]$$

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.

Atoms

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

$$A, B ::= p(t_1, \dots, t_m) \quad , p/m \in \Pi$$
$$\left[\quad \mid \quad (t \approx t') \quad \text{(equation)} \quad \right]$$

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.

Many-sorted case:

If $a(p) = s_1 \dots s_m$, we require that t_i is a term of sort s_i for $i = 1, \dots, m$.

Literals

$L ::= A$ (positive literal)
| $\neg A$ (negative literal)

Clauses

$C, D ::= \perp$ (empty clause)
| $L_1 \vee \dots \vee L_k, k \geq 1$ (non-empty clause)

General First-Order Formulas

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

F, G, H	$::=$	\perp	(falsum)
		\top	(verum)
		A	(atomic formula)
		$\neg F$	(negation)
		$(F \wedge G)$	(conjunction)
		$(F \vee G)$	(disjunction)
		$(F \rightarrow G)$	(implication)
		$(F \leftrightarrow G)$	(equivalence)
		$\forall xF$	(universal quantification)
		$\exists xF$	(existential quantification)

Notational Conventions

We omit brackets according to the following rules:

- $\neg >_p \wedge >_p \vee >_p \rightarrow >_p \leftrightarrow$
(binding precedences)
- \vee and \wedge are associative and commutative
- \rightarrow is right-associative

$Q_{x_1, \dots, x_n} F$ abbreviates $Q_{x_1} \dots Q_{x_n} F$.

Notational Conventions

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

Examples:

$$s + t * u \quad \text{for} \quad +(s, *(t, u))$$

$$s * u \leq t + v \quad \text{for} \quad \leq (*(s, u), +(t, v))$$

$$-s \quad \text{for} \quad -(s)$$

$$0 \quad \text{for} \quad 0()$$

Conventions

In what follows we will use the following conventions:

constants (0-ary function symbols) are denoted with a, b, c, d, \dots

function symbols with arity ≥ 1 are denoted

- f, g, h, \dots if the formulae are interpreted into arbitrary algebras
- $+, -, s, \dots$ if the intended interpretation is into numerical domains

predicate symbols with arity 0 are denoted P, Q, R, S, \dots

predicate symbols with arity ≥ 1 are denoted

- p, q, r, \dots if the formulae are interpreted into arbitrary algebras
- $\leq, \geq, <, >$ if the intended interpretation is into numerical domains

variables are denoted x, y, z, \dots

Example: Peano Arithmetic

Signature:

$$\Sigma_{PA} = (\Omega_{PA}, \Pi_{PA})$$

$$\Omega_{PA} = \{0/0, +/2, */2, s/1\}$$

$$\Pi_{PA} = \{\leq /2, < /2\}$$

$$+, *, <, \leq \text{ infix; } * >_p + >_p < >_p \leq$$

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.

Example: Specifying LISP lists

Signature:

$$\Sigma_{\text{Lists}} = (\Omega_{\text{Lists}}, \Pi_{\text{Lists}})$$

$$\Omega_{\text{Lists}} = \{\text{car}/1, \text{cdr}/1, \text{cons}/2\}$$

$$\Pi_{\text{Lists}} = \emptyset$$

Examples of formulae:

$$\forall x, y \quad \text{car}(\text{cons}(x, y)) \approx x$$

$$\forall x, y \quad \text{cdr}(\text{cons}(x, y)) \approx y$$

$$\forall x \quad \text{cons}(\text{car}(x), \text{cdr}(x)) \approx x$$

Many-sorted signatures

Example:

Signature

$$S = \{\text{array}, \text{index}, \text{element}\}$$

set of sorts

$$\Omega = \{\text{read}, \text{write}\}$$

$$a(\text{read}) = \text{array} \times \text{index} \rightarrow \text{element}$$

$$a(\text{write}) = \text{array} \times \text{index} \times \text{element} \rightarrow \text{array}$$

$$\Pi = \emptyset$$

$$X = \{X_s \mid s \in S\}$$

Examples of formulae:

$$\forall x : \text{array} \quad \forall i : \text{index} \quad \forall j : \text{index} \quad (i \approx j \rightarrow \text{write}(x, i, \text{read}(x, j)) \approx x)$$

$$\forall x : \text{array} \quad \forall y : \text{array} \quad (x \approx y \leftrightarrow \forall i : \text{index} \quad (\text{read}(x, i) \approx \text{read}(y, i)))$$

Bound and Free Variables

In $Qx F$, $Q \in \{\exists, \forall\}$, 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:

$$\forall y \quad (\forall x \quad p(x) \rightarrow q(x, y))$$

The diagram illustrates the scope of variables in the expression $\forall y \quad (\forall x \quad p(x) \rightarrow q(x, y))$. A large curly brace labeled "scope" spans the entire expression, indicating that y is bound. A smaller curly brace labeled "scope" spans the sub-expression $(\forall x \quad p(x) \rightarrow q(x, y))$, indicating that x is bound within this sub-expression. The variable y is colored red, x is colored blue, and the x and y in $q(x, y)$ are colored green and red respectively.

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

$$\sigma : X \rightarrow T_{\Sigma}(X)$$

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***(σ).

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

$$\sigma : X \rightarrow T_{\Sigma}(X)$$

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***(σ).

Many-sorted case: Substitutions must be sort-preserving:

If x is a variable of sort s , then $\sigma(x)$ must be a term of sort s .

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 .

Application of a Substitution

“Homomorphic” extension of σ to terms and formulas:

$$f(s_1, \dots, s_n)\sigma = f(s_1\sigma, \dots, s_n\sigma)$$

$$\perp\sigma = \perp$$

$$\top\sigma = \top$$

$$p(s_1, \dots, s_n)\sigma = p(s_1\sigma, \dots, s_n\sigma)$$

$$(u \approx v)\sigma = (u\sigma \approx v\sigma)$$

$$\neg F\sigma = \neg(F\sigma)$$

$$(F\rho G)\sigma = (F\sigma\rho G\sigma) ; \text{ for each binary connective } \rho$$

$$(Qx F)\sigma = Qz (F [x \mapsto z]\sigma) ; \text{ with } z \text{ a fresh variable}$$

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

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

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 $\Sigma\text{-Alg}$ we denote the class of all Σ -algebras.

Structures

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

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

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.

A **many-sorted Σ -algebra** (also called Σ -interpretation or Σ -structure),

where $\Sigma = (S, \Omega, \Pi)$ is a triple

$$\mathcal{A} = ((\{U_s\}_{s \in S}, (f_{\mathcal{A}} : U_{s_1} \times \dots \times U_{s_n} \rightarrow U_s)_{\substack{f \in \Omega, \\ a(f) = s_1 \dots s_n \rightarrow s}}, (p_{\mathcal{A}} : U_{s_1} \times \dots \times U_{s_m} \rightarrow \{0, 1\})_{\substack{p \in \Pi \\ a(p) = s_1 \dots s_m}}))$$

where $U_s \neq \emptyset$ is a set, called the **universe** of \mathcal{A} of sort s .

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 $\beta : X \rightarrow \mathcal{A}$.

Variable assignments are the semantic counterparts of substitutions.

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 $\beta : X \rightarrow \mathcal{A}$.

Variable assignments are the semantic counterparts of substitutions.

Many-sorted case:

$$\beta = \{\beta_s\}_{s \in S}, \beta_s : X_s \rightarrow U_s$$

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

By structural induction we define

$$\mathcal{A}(\beta) : T_{\Sigma}(X) \rightarrow \mathcal{A}$$

as follows:

$$\mathcal{A}(\beta)(x) = \beta(x), \quad x \in X$$

$$\mathcal{A}(\beta)(f(s_1, \dots, s_n)) = f_{\mathcal{A}}(\mathcal{A}(\beta)(s_1), \dots, \mathcal{A}(\beta)(s_n)), \quad f/n \in \Omega$$

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 $\beta[x \mapsto a] : X \rightarrow \mathcal{A}$, 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

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

$$\mathcal{A}(\beta)(\perp) = 0$$

$$\mathcal{A}(\beta)(\top) = 1$$

$$\mathcal{A}(\beta)(p(s_1, \dots, s_n)) = 1 \iff (\mathcal{A}(\beta)(s_1), \dots, \mathcal{A}(\beta)(s_n)) \in p_{\mathcal{A}}$$

$$\mathcal{A}(\beta)(s \approx t) = 1 \iff \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t)$$

$$\mathcal{A}(\beta)(\neg F) = 1 \iff \mathcal{A}(\beta)(F) = 0$$

$$\mathcal{A}(\beta)(F \rho G) = B_{\rho}(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G))$$

with B_{ρ} the Boolean function associated with ρ

$$\mathcal{A}(\beta)(\forall x F) = \min_{a \in U} \{ \mathcal{A}(\beta[x \mapsto a])(F) \}$$

$$\mathcal{A}(\beta)(\exists x F) = \max_{a \in U} \{ \mathcal{A}(\beta[x \mapsto a])(F) \}$$

Example

The “Standard” Interpretation for Peano Arithmetic:

$$U_{\mathbb{N}} = \{0, 1, 2, \dots\}$$

$$0_{\mathbb{N}} = 0$$

$$s_{\mathbb{N}} : n \mapsto n + 1$$

$$+_{\mathbb{N}} : (n, m) \mapsto n + m$$

$$*_{\mathbb{N}} : (n, m) \mapsto n * m$$

$$\leq_{\mathbb{N}} = \{(n, m) \mid n \text{ less than or equal to } m\}$$

$$<_{\mathbb{N}} = \{(n, m) \mid n \text{ less than } m\}$$

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

$$\mathbb{N}(\beta)(s(x) + s(0)) = 3$$

$$\mathbb{N}(\beta)(x + y \approx s(y)) = 1$$

$$\mathbb{N}(\beta)(\forall x, y(x + y \approx y + x)) = 1$$

$$\mathbb{N}(\beta)(\forall z z \leq y) = 0$$

$$\mathbb{N}(\beta)(\forall x \exists y x < y) = 1$$