

Example: Peano Arithmetic

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

Examples of formulas over this signature are:

$$\begin{aligned}\forall x, y ((x < y \vee x \approx y) \leftrightarrow \exists z (x + z \approx y)) \\ \exists x \forall y (x + y \approx y) \\ \forall x, y (x * s(y) \approx x * y + x) \\ \forall x, y (s(x) \approx s(y) \rightarrow x \approx y) \\ \forall x \exists y (x < y \wedge \neg \exists z (x < z \wedge z < y))\end{aligned}$$

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):

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

The prefix order \leq , 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 $Qx F$, $Q \in \{\exists, \forall\}$, 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*.

Example:

$$\forall y \left(\overbrace{((\forall x \underbrace{P(x)}_{\text{scope of } x}) \rightarrow R(x, y))}_{\text{scope of } y} \right)$$

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.

Substitutions are mappings

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

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

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

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

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

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

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

The *modification* of a substitution 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:

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

$$\perp\sigma = \perp$$

$$\top\sigma = \top$$

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

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

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

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

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

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

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

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

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

Variable assignments are the semantic counterparts of substitutions.

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

By structural induction we define

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

as follows:

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

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

$$\begin{aligned}
 \mathcal{A}(\beta)(\perp) &= 0 \\
 \mathcal{A}(\beta)(\top) &= 1 \\
 \mathcal{A}(\beta)(P(s_1, \dots, s_n)) &= \text{if } (\mathcal{A}(\beta)(s_1), \dots, \mathcal{A}(\beta)(s_n)) \in P_{\mathcal{A}} \text{ then } 1 \text{ else } 0 \\
 \mathcal{A}(\beta)(s \approx t) &= \text{if } \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t) \text{ then } 1 \text{ else } 0 \\
 \mathcal{A}(\beta)(\neg F) &= 1 - \mathcal{A}(\beta)(F) \\
 \mathcal{A}(\beta)(F \wedge G) &= \min(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
 \mathcal{A}(\beta)(F \vee G) &= \max(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
 \mathcal{A}(\beta)(F \rightarrow G) &= \max(1 - \mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
 \mathcal{A}(\beta)(F \leftrightarrow G) &= \text{if } \mathcal{A}(\beta)(F) = \mathcal{A}(\beta)(G) \text{ then } 1 \text{ else } 0 \\
 \mathcal{A}(\beta)(\forall x F) &= \min_{a \in U_{\mathcal{A}}} \{ \mathcal{A}(\beta[x \mapsto a])(F) \} \\
 \mathcal{A}(\beta)(\exists x F) &= \max_{a \in U_{\mathcal{A}}} \{ \mathcal{A}(\beta[x \mapsto a])(F) \}
 \end{aligned}$$

Example

The “Standard” Interpretation for Peano Arithmetic:

$$\begin{aligned}
 U_{\mathbb{N}} &= \{0, 1, 2, \dots\} \\
 0_{\mathbb{N}} &= 0 \\
 s_{\mathbb{N}} &: n \mapsto n + 1 \\
 +_{\mathbb{N}} &: (n, m) \mapsto n + m \\
 *_{\mathbb{N}} &: (n, m) \mapsto n * m \\
 <_{\mathbb{N}} &= \{ (n, m) \mid n \text{ less than } m \}
 \end{aligned}$$

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

Values over \mathbb{N} for sample terms and formulas:

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

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

Ground Terms and Closed Formulas

If t is a ground term, then $\mathcal{A}(\beta)(t)$ does not depend on β :

$$\mathcal{A}(\beta)(t) = \mathcal{A}(\beta')(t)$$

for every β and β' .

Analogously, if F is a closed formula, then $\mathcal{A}(\beta)(F)$ does not depend on β :

$$\mathcal{A}(\beta)(F) = \mathcal{A}(\beta')(F)$$

for every β and β' .

An element $a \in U_{\mathcal{A}}$ is called *term-generated*, if $a = \mathcal{A}(\beta)(t)$ for some ground term t .

In general, not every element of an algebra is term-generated.

3.3 Models, Validity, and Satisfiability

F is *true* in \mathcal{A} under assignment β :

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

F is *true* in \mathcal{A} (\mathcal{A} is a *model* of F ; F is *valid* in \mathcal{A}):

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

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

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

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 $\mathcal{A} \in \Sigma\text{-Alg}$ and $\beta \in X \rightarrow U_{\mathcal{A}}$, whenever $\mathcal{A}, \beta \models F$, then $\mathcal{A}, \beta \models G$.

F and G are called equivalent, written $F \models\!\!\!\models G$, if for all $\mathcal{A} \in \Sigma\text{-Alg}$ and $\beta \in X \rightarrow U_{\mathcal{A}}$ we have $\mathcal{A}, \beta \models F \Leftrightarrow \mathcal{A}, \beta \models G$.

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., $N \models F$

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma\text{-Alg}$ 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 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 $F \wedge \neg G$ 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.

3.4 Algorithmic Problems

Validity(F): $\models F$?

Satisfiability(F): F satisfiable?

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

Model(\mathcal{A}, F): $\mathcal{A} \models F$?

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

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

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

Theory of an Algebra

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

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

Problem of axiomatizability:

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

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

(analogously for classes of algebras).

Two Interesting Theories

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

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

(Non-)Computability Results

1. For most signatures Σ , validity is undecidable for Σ -formulas.
(One can easily encode Turing machines in most signatures.)
2. Gödel's completeness theorem:
For each signature Σ , the set of valid Σ -formulas is recursively enumerable.
(We will prove this by giving complete deduction systems.)
3. Gödel's incompleteness theorem:
For $\Sigma = \Sigma_{PA}$ and $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *, <)$, the theory $\text{Th}(\mathbb{N}_*)$ is not recursively enumerable.

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

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 exponential time and PSPACE-complete.

3.5 Normal Forms and Skolemization

Study of normal forms motivated by

- reduction of logical concepts,
- efficient data structures for theorem proving.

The main problem in first-order logic is the treatment of quantifiers. The subsequent normal form transformations are intended to eliminate many of them.

Prenex Normal Form (Traditional)

Prenex formulas have the form

$$\mathbf{Q}_1 x_1 \dots \mathbf{Q}_n x_n F,$$

where F is quantifier-free and $\mathbf{Q}_i \in \{\forall, \exists\}$; we call $\mathbf{Q}_1 x_1 \dots \mathbf{Q}_n x_n$ the *quantifier prefix* and F the *matrix* of the formula.

Computing prenex normal form by the reduction system \Rightarrow_P :

$$\begin{aligned} H[(F \leftrightarrow G)]_p &\Rightarrow_P H[(F \rightarrow G) \wedge (G \rightarrow F)]_p \\ H[\neg \mathbf{Q}x F]_p &\Rightarrow_P H[\bar{\mathbf{Q}}x \neg F]_p \\ H[((\mathbf{Q}x F) \circ G)]_p &\Rightarrow_P H[\mathbf{Q}y (F\{x \mapsto y\} \circ G)]_p, \\ &\quad \circ \in \{\wedge, \vee\} \\ H[((\mathbf{Q}x F) \rightarrow G)]_p &\Rightarrow_P H[\bar{\mathbf{Q}}y (F\{x \mapsto y\} \rightarrow G)]_p, \\ H[(F \circ (\mathbf{Q}x G))]_p &\Rightarrow_P H[\mathbf{Q}y (F \circ G\{x \mapsto y\})]_p, \\ &\quad \circ \in \{\wedge, \vee, \rightarrow\} \end{aligned}$$

Here y is always assumed to be some fresh variable and $\bar{\mathbf{Q}}$ denotes the quantifier *dual* to \mathbf{Q} , i. e., $\bar{\forall} = \exists$ and $\bar{\exists} = \forall$.

Skolemization

Intuition: replacement of $\exists y$ by a concrete choice function computing y from all the arguments y depends on.

Transformation \Rightarrow_S

(to be applied outermost, *not* in subformulas):

$$\forall x_1, \dots, x_n \exists y F \Rightarrow_S \forall x_1, \dots, x_n F\{y \mapsto f(x_1, \dots, x_n)\}$$

where f/n is a new function symbol (*Skolem function*).

Together: $F \Rightarrow_P^* \underbrace{G}_{\text{prenex}} \Rightarrow_S^* \underbrace{H}_{\text{prenex, no } \exists}$

Theorem 3.7 Let F , G , and H as defined above and closed. Then

- (i) F and G are equivalent.
- (ii) $H \models G$ but the converse is not true in general.
- (iii) G satisfiable (w. r. t. Σ -Alg) $\Leftrightarrow H$ satisfiable (w. r. t. Σ' -Alg) where $\Sigma' = (\Omega \cup SKF, \Pi)$ if $\Sigma = (\Omega, \Pi)$.

The Complete Picture

$$\begin{aligned}
 F &\Rightarrow_P^* Q_1 y_1 \dots Q_n y_n G && (G \text{ quantifier-free}) \\
 &\Rightarrow_S^* \forall x_1, \dots, x_m H && (m \leq n, H \text{ quantifier-free}) \\
 &\Rightarrow_{CNF}^* \underbrace{\underbrace{\forall x_1, \dots, x_m}_{\text{leave out}} \bigwedge_{i=1}^k \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}}_{F'}
 \end{aligned}$$

$N = \{C_1, \dots, C_k\}$ is called the *clausal (normal) form (CNF)* of F .

Note: The variables in the clauses are implicitly universally quantified.

Theorem 3.8 *Let F be closed. Then $F' \models F$. (The converse is not true in general.)*

Theorem 3.9 *Let F be closed. Then F is satisfiable iff F' is satisfiable iff N is satisfiable*

Optimization

The normal form algorithm described so far leaves lots of room for optimization. Note that we only can preserve satisfiability anyway due to Skolemization.

- the size of the CNF is exponential when done naively; the transformations we introduced already for propositional logic avoid this exponential growth;
- we want to preserve the original formula structure;
- we want small arity of Skolem functions (see next section).

3.6 Getting Skolem Functions with Small Arity

A clause set that is better suited for automated theorem proving can be obtained using the following steps:

- eliminate trivial subformulas
- replace beneficial subformulas
- produce a negation normal form (NNF)
- apply miniscoping
- rename all variables
- Skolemize
- push quantifiers upward
- apply distributivity

We start with a closed formula.

Elimination of Trivial Subformulas

Eliminate subformulas \top and \perp essentially as in the propositional case modulo associativity/commutativity of \wedge , \vee :

$$\begin{aligned}
 H[(F \wedge \top)]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
 H[(F \vee \perp)]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
 H[(F \leftrightarrow \perp)]_p &\Rightarrow_{\text{OCNF}} H[\neg F]_p \\
 H[(F \leftrightarrow \top)]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
 H[(F \vee \top)]_p &\Rightarrow_{\text{OCNF}} H[\top]_p \\
 H[(F \wedge \perp)]_p &\Rightarrow_{\text{OCNF}} H[\perp]_p \\
 H[\neg \top]_p &\Rightarrow_{\text{OCNF}} H[\perp]_p \\
 H[\neg \perp]_p &\Rightarrow_{\text{OCNF}} H[\top]_p \\
 H[(F \rightarrow \perp)]_p &\Rightarrow_{\text{OCNF}} H[\neg F]_p \\
 H[(F \rightarrow \top)]_p &\Rightarrow_{\text{OCNF}} H[\top]_p \\
 H[(\perp \rightarrow F)]_p &\Rightarrow_{\text{OCNF}} H[\top]_p \\
 H[(\top \rightarrow F)]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
 H[\text{Q}x \top]_p &\Rightarrow_{\text{OCNF}} H[\top]_p \\
 H[\text{Q}x \perp]_p &\Rightarrow_{\text{OCNF}} H[\perp]_p
 \end{aligned}$$