Chapter 2
Basic Notions

This chapter provides a brief introduction to basic notions and definitions from algebra and predicate logic. For further discussion, examples, and presentation of concepts see e.g. [55, 22, 169, 106].

2.1 Signatures and $\Sigma$-structures

A signature constitutes the syntax of a language, i.e. the symbols used to compose language expressions like terms and constraints. Their interpretation or semantics is defined by means of an appropriate structure.

Definition 1 (signature) A (many-sorted) signature $\Sigma = (S, F, R)$ is defined by a set $S$ of sorts, a set $F$ of function symbols, and a set $R$ of predicate symbols. The sets $S$, $F$, and $R$ are mutually disjoint.

Every function symbol $f \in F$ and every predicate symbol $r \in R$ is associated with a declaration $f : s_1 \ldots s_n \rightarrow s$ and $r : s_1 \ldots s_m$, $s, s_i \in S$, $n, m \geq 0$, and thus with an arity $n$ or $m$, resp. A symbol $f$ with $n = 0$ is a constant symbol.

Let $X^s$ be a set of variables of sort $s \in S$. A set of $\Sigma$-variables is a set $X = \bigcup_{s \in S} X^s$, where the sets $X^s$ are non-empty and mutually disjoint.

A $\Sigma$-structure builds on a signature $\Sigma$ and defines the semantics of the symbols of $\Sigma$.

Definition 2 ($\Sigma$-structure) Let $\Sigma = (S, F, R)$ be a signature. A $\Sigma$-structure $D = (\{D^s | s \in S\}, \{f^D | f \in F\}, \{r^D | r \in R\})$ consists of an $S$-sorted set of non-empty carrier sets $D^s$ with $s \in S$, a set of functions $f^D$ with $f \in F$, and a set of predicates $r^D$ with $r \in R$.

For a function symbol $f \in F$ with $f : s_1 \ldots s_n \rightarrow s$ let $f^D$ be an $n$-ary function, such that $f^D : D^{s_1} \times \ldots \times D^{s_n} \rightarrow D^s$ holds.

For a predicate symbol $r \in R$ with $r : s_1 \ldots s_m$ let $r^D$ be a $m$-ary predicate, such that $r^D \subseteq D^{s_1} \times \ldots \times D^{s_m}$ holds.
The following example illustrates these definitions and will be used in the subsequent sections.

**Example 1** Let \( \Sigma_N = (S, F, R) \) be a signature consisting of the set of sorts \( S = \{ \text{nat} \} \), the set \( F = \{ \text{succ}, \text{plus}, \text{mul}, 0, 1, 2, \ldots \} \) of function and constant symbols and the predicate symbols \( R = \{ \text{eq}, \text{geq} \} \) with the following declarations:

\[
\begin{align*}
\text{succ} : & \text{nat} \to \text{nat} \\
\text{plus}, \text{mul} : & \text{nat nat} \to \text{nat} \\
0, 1, 2, \ldots : & \text{nat} \\
\text{eq}, \text{geq} : & \text{nat nat}
\end{align*}
\]

We define a \( \Sigma_N \)-structure \( \mathcal{D}_N \) by \( \mathcal{D}_N = (\{N\}, \{f^N | f \in F\}, \{r^N | r \in R\}) \), where the carrier-set \( N \) is the set of natural numbers on which the functions \( f^N \) and predicates \( r^N \) apply, e.g.

\[
\begin{align*}
\text{succ}^N : & N \to N \text{ and for every } x \in N \text{ holds: } \text{succ}^N(x) = (x + 1), \\
\text{plus}^N : & N \times N \to N, \text{ and for every } x, y \in N \text{ holds: } \text{plus}^N(x, y) = (x + y), \\
0^N : & N \text{ with } 0^N = 0, \\
1^N : & N \text{ with } 1^N = 1, \\
\text{eq}^N \subseteq & N \times N, \text{ where for all } x, y \in N \text{ holds: } \text{eq}^N(x, y) \text{ iff } x = y, \\
\text{geq}^N \subseteq & N \times N, \text{ where for all } x, y \in N \text{ holds: } \text{geq}^N(x, y) \text{ iff } x \geq y.
\end{align*}
\]

\[\Box\]

### 2.2 Terms, formulae, and validity

Based on the notion of a signature, we define terms and formulae. We provide these syntactic elements with a meaning, i.e. a semantics, and determine the validity of formulae with the help of the corresponding \( \Sigma \)-structure.

In the following, let \( \Sigma = (S, F, R) \) be a signature, let \( X \) be a set of \( \Sigma \)-variables, and let \( \mathcal{D} \) be a \( \Sigma \)-structure.

Terms are built from the symbols of \( \Sigma \). They are defined inductively. Besides variables and constant symbols, there are terms composed of subterms based on the declarations of the involved function symbols.

**Definition 3** (term, ground term) The set \( \mathcal{T}(F, X) \) of *terms* over \( \Sigma \) and \( X \) is defined as follows: \( \mathcal{T}(F, X) = \bigcup_{s \in S} \mathcal{T}(F, X)^s \), where for every sort \( s \in S \) the set \( \mathcal{T}(F, X)^s \) of terms of sort \( s \) is the smallest set containing

1. every variable \( x \in X^s \) (of sort \( s \)),
2. every 0-ary function symbol \( f \in F \) with \( f : s \), i.e. every constant symbol, and
3. every expression \( f(t_1, \ldots, t_n) \), \( n \geq 1 \), where \( f \in F \) is a function symbol with 

declaration \( f : s_1 \ldots s_n \to s \) and every \( t_i, i \in \{1, \ldots, n\} \), is a (composite) 
term of \( T(F, X)^{s_i} \).

Terms without variables are **ground terms**.

A position \( p \) in a term \( t \) is represented by a sequence of natural numbers. 
The empty sequence is denoted by \( \epsilon \). We recursively define \( t[p] \) to denote the 
subterm of \( t \) at position \( p \) as \( t[\epsilon] = t \) and \( f(t_1, \ldots, t_n)[i, p] = t[i][p] \). By \( t[r]_p \) we 
denote the term which is obtained from \( t \) as the result of the replacement of 
the subterm \( t[p] \) with the term \( r \).

**Example 2** Let \( x, y, z \in X \). For our signature \( \Sigma_N \) from above, \( x, 2, \text{succ}(x), \text{plus}(2, \text{succ}(3)) \), and \( \text{plus}(\text{succ}(x), \text{mul}(2, \text{succ}(y))) \) are terms.

For a term \( t = \text{plus}(x, \text{mul}(2, \text{succ}(y))) \) examples of subterms are \( t[\epsilon] = t \), 
\( t[1] = x \), \( t[2] = \text{mul}(2, \text{succ}(y)) \), and \( t[221] = y \), and replacements are given by e.g. 
\( t[\text{mul}(2, z)]_2 = \text{plus}(x, \text{mul}(2, z)) \) and \( t[1][221] = \text{plus}(x, \text{mul}(2, \text{succ}(1))) \).  

Terms represent elements or objects of the corresponding domain. For example, terms over \( \Sigma_N \) are arithmetic expressions. Similarly, boolean expressions 
can be built over an appropriate signature \( \Sigma_B \).

To determine the semantics of terms w.r.t. a \( \Sigma \)-structure \( D \) we must assign 
values to the variables of the terms. This is done by means of a valuation.

**Definition 4** (valuation) An \( S \)-sorted family of mappings \( \varsigma : X \to D = (\varsigma^s : X^s \to D^s)^{s \in S} \) which assigns each variable \( x \in X^s \) an element of the carrier 
set \( D^s \), \( s \in S \), is a **valuation**.

Now, we can evaluate terms w.r.t. a structure \( D \) and a valuation \( \varsigma \).

**Definition 5** (evaluation of terms) Let \( \varsigma : X \to D \) be a valuation. The **evaluation** \( \varsigma : T(F, X) \to D \) of a term w.r.t. the structure \( D \) and the valuation 
\( \varsigma \) is a family of mappings \((\varsigma^s : T(F, X)^s \to D^s)^{s \in S}\) with:

- \( \varsigma^s(x) = \varsigma^s(x) \) for every variable \( x \) of sort \( s \),
- \( \varsigma^s(f) = f^D \) for every constant symbol \( f \in F \) with \( f : s \), and
- \( \varsigma^s(f(t_1, \ldots, t_n)) = f^D(\varsigma^{s_1}(t_1), \ldots, \varsigma^{s_n}(t_n)) \) for every function symbol \( f \in F \) 
  with \( f : s_1 \ldots s_n \to s \) and every sort \( s_1, \ldots, s_n, s \in S \) and all terms 
  \( t_i \in T(F, X)^{s_i}, i \in \{1, \ldots, n\} \).

The evaluation of a variable is just its valuation, the evaluation of a constant 
symbol is the corresponding constant from the structure. For a composite 
term we evaluate its subterms and apply the corresponding function from the 
structure.

The set \( \text{Formulae}(\Sigma, X) \) of formulae of (first-order) predicate logic deter-
mines the **syntax of predicate logic**.

**Definition 6** (formulae of predicate logic) The set of **formulae of pre-
dicate logic** over a signature \( \Sigma \) and a set of variables \( X \), denoted by 
\( \text{Formulae}(\Sigma, X) \), is inductively defined as follows:
1. For all predicate symbols \( r : s_1 \ldots s_m \) and all terms \( t_i \in T(F, X)^{s_i} \), \( i \in \{1, \ldots, m\} \), the expression \( r(t_1, \ldots, t_m) \) is a (atomic) formula.

2. \( \text{true} \) and \( \text{false} \) are (atomic) formulae.

3. For every formula \( \phi \) the expression \( \neg \phi \) is a formula.

4. For all formulae \( \phi \) and \( \psi \) the following expressions are formulae too:
   \( (\phi \lor \psi) \), \( (\phi \land \psi) \), \( (\phi \rightarrow \psi) \), and \( (\phi \iff \psi) \).

5. If \( \phi \) is a formula and \( x \in X \) is a variable, then \( (\forall x. \phi) \) and \( (\exists x. \phi) \) are formulae.

We denote the set of variables occurring in a term or formula \( F \), resp., by \( \text{var}(F) \). The quantifiers \( \forall \) and \( \exists \) bind variables in formulae. We introduce certain notions concerning quantifiers.

**Definition 7** (bound and free variable, open and closed formula) An occurrence of a variable \( x \) in a formula \( \phi \in \text{Formulae}(\Sigma, X) \) is called **bound**, if \( x \) appears in a subformula of \( \phi \) in the form \( \exists x. \psi \) or \( \forall x. \psi \). Otherwise \( x \) is a **free** variable. A formula \( \phi \) without occurrences of free variables is called a **closed** formula, otherwise \( \phi \) is **open**.

**Definition 8** (universal and existential closure) Let \( \{x_1, \ldots, x_n\} \subseteq X \) be the set of free variables of a predicate logic formula \( \phi \in \text{Formulae}(\Sigma, X) \).

The **universal closure** \( \forall \phi \) and the **existential closure** \( \exists \phi \) of \( \phi \) are defined by

\[
\forall \phi = \forall x_1 \ldots \forall x_n. \phi \quad \text{and} \quad \exists \phi = \exists x_1 \ldots \exists x_n. \phi, \text{ resp.}
\]

The expression \( \bar{Y} \) with \( Y \subseteq X \) denotes a (arbitrary) sequence of the variables of the set \( Y \). By \( \exists_{\bar{Y}} \psi \) we denote the existential closure of the formula \( \psi \) except for the variables of \( Y \).

In the following, we write \( \forall x, y. \phi \) instead of \( \forall x. \forall y. \phi \) and \( \exists x, y. \phi \) instead of \( \exists x. \exists y. \phi \) as is usually done.

**Example 3** Consider the signature \( \Sigma_N \) and the structure \( D_N \) from Example 1 and the variables \( \{x, y, z\} \subseteq X \).

The following formulae are elements of \( \text{Formulae}(\Sigma_N, X) \):
\( \text{true}, \text{false}, \text{geq}(2, x), \text{eq}(\text{mul}(2, 2), 4), \neg \text{geq}(2, x) \lor \neg \text{geq}(x, 2), \text{true} \rightarrow \text{false}, \text{eq}(2, x) \iff \text{eq}(\text{succ}(2), \text{succ}(x)), \forall x, y. \text{eq}(z, \text{plus}(x, y)), \forall x. \exists y. \text{eq}(x, \text{succ}(y)), \text{geq}(x, 2) \rightarrow \exists x. \text{eq}(x, 2) \).

Consider \( p = \forall x, y. \text{eq}(z, \text{plus}(x, y)) \) and \( q = \text{geq}(x, 2) \rightarrow \exists x. \text{eq}(x, 2) \). The variables \( x \) and \( y \) are bound in formula \( p \), while the variable \( z \) is free. In the formula \( q \) the first occurrence of variable \( x \) is free, while its second occurrence is bound by the existential quantifier \( \exists \).

The formulae \( \text{true}, \text{false}, \text{eq}(\text{mul}(2, 2), 4), \forall x. \exists y. \text{eq}(x, \text{succ}(y)), \) and \( \text{true} \rightarrow \text{false} \) are closed, all other formulae given above are open.

Let \( Y = \{x, y\} \subseteq \{x, y, z\} \subseteq X \). The following holds:
\( \exists_{\bar{Y}} \text{eq}(z, \text{plus}(x, y)) = \exists_{x, y} \text{eq}(z, \text{plus}(x, y)) = \exists z. \text{eq}(z, \text{plus}(x, y)) \).
2.2 Terms, formulae, and validity

The semantics of predicate logic is determined by the assignment of a meaning to every formula w.r.t. the associated structure. We define the validity relation between structures and formulae (see e.g. [55]).

**Definition 9** (validity, ⊨, model) Let \( \phi, \psi \in \text{Formulae}(\Sigma, X) \) be formulae of predicate logic. Let \( \varsigma : X \to D \) be a valuation. A valuation which maps the variable \( x \in X \) to \( a \in D \) and all other variables \( y \) to \( \varsigma(y) \) is denoted by \( \varsigma[x/a] : X \to D \), i.e.

\[
\varsigma[x/a](y) = \begin{cases} 
\varsigma(y) & \text{if } y \neq x, \\
 a & \text{otherwise}.
\end{cases}
\]

The relation \( \models \) is defined as follows:

\[
(D, \varsigma) \models r(t_1, \ldots, t_m) \text{ if and only if } (\varsigma(t_1), \ldots, \varsigma(t_m)) \in r^D,
\]

\[
(D, \varsigma) \models \text{true},
\]

\[
(D, \varsigma) \nvdash \phi \quad \text{iff} \quad (D, \varsigma) \not\models \phi,
\]

\[
(D, \varsigma) \models \phi \land \psi \quad \text{iff} \quad (D, \varsigma) \models \phi \text{ and } (D, \varsigma) \models \psi,
\]

\[
(D, \varsigma) \models \phi \lor \psi \quad \text{iff} \quad (D, \varsigma) \models \phi \text{ or } (D, \varsigma) \models \psi,
\]

\[
(D, \varsigma) \models \phi \rightarrow \psi \quad \text{iff} \quad (D, \varsigma) \nvdash \phi \text{ or } (D, \varsigma) \models \psi,
\]

\[
(D, \varsigma) \models \phi \leftrightarrow \psi \quad \text{iff} \quad (D, \varsigma) \models \phi \text{ and } (D, \varsigma) \models \psi \text{ and } (D, \varsigma) \models \psi \rightarrow \phi,
\]

\[
(D, \varsigma) \models \forall x. \phi \quad \text{iff} \quad (D, \varsigma[x/a]) \models \phi \text{ for every } a \in D^s, s \in S, x \in X^s,
\]

\[
(D, \varsigma) \models \exists x. \phi \quad \text{iff} \quad (D, \varsigma[x/a]) \models \phi \text{ for at least one } a \in D^s, s \in S, x \in X^s.
\]

A formula \( \phi \) is **valid** in \( D \), i.e. it holds \( D \models \phi \), if for every valuation \( \varsigma : X \to D \) holds: \( (D, \varsigma) \models \phi \). In this case, we call \( D \) a **model** of \( \phi \).

**Example 4** Consider the signature \( \Sigma_N \) and the structure \( D_N \) of Example 1. Let \( \varsigma \) be a valuation with \( \varsigma(x) = 1 \), \( \varsigma(y) = 2 \), and \( \varsigma(z) = 3 \). We study the validity of various formulae:

\[
(D_N, \varsigma) \models \text{true} \quad \text{and} \quad (D_N, \varsigma) \nvdash \text{false}.
\]

\[
(D_N, \varsigma) \models \text{geq}(2, x), \text{ because } 2^N = 2 \geq 1 = 1^N.
\]

\[
(D_N, \varsigma) \models \text{eq}(\text{plus}(2, 2), 4), \text{ because } \text{plus}^N(2, 2) = 4.
\]

\[
(D_N, \varsigma) \nvdash \text{geq}(2, x) \lor \neg \text{eq}(x, 2),
\]

because the validity of one subformula \( \phi \) or \( \psi \) of a formula \( \phi \lor \psi \) is sufficient and \( (1, 2) \notin \text{eq}^N \) resp. \( 1 \nleq 2 \).

\[
(D_N, \varsigma) \nvdash \text{true} \longrightarrow \text{false}
\]

according to the definition of validity of formulae of the form \( \phi \longrightarrow \psi \) (see above).

\[
(D_N, \varsigma) \models (\text{eq}(2, x) \longleftrightarrow \text{eq}(\text{succ}(2), \text{succ}(x))),
\]

because \( (2, 1) \notin \text{eq}^N \) resp. \( 2 \neq 1 \) and \( (3, 2) \notin \text{eq}^N \) resp. \( 3 \neq 2 \).

\[
(D_N, \varsigma) \nvdash \forall x, y. \text{eq}(z, \text{plus}(x, y)),
\]

because there are valuations \( \varsigma' \) of \( x \) and \( y \) such that \( 3 \neq \varsigma'(x) + \varsigma'(y) \).

\[
(D_N, \varsigma) \nvdash \forall x. \exists y. \text{eq}(x, \text{succ}(y)),
\]
because when \( x \) has the value 0 there is no value for \( y \) such that \( x = y + 1 \).

\[(\mathcal{D}_\mathbb{N}, \varsigma) \models \text{geq}(x, 2) \rightarrow \exists x. \text{eq}(x, 2), \text{because} \ (1, 2) \not\in \text{geq}^\mathbb{N}.\]

Of the above formulae the following are valid in \( \mathcal{D}_\mathbb{N} \), i.e. they hold in \( \mathcal{D}_\mathbb{N} \) for every valuation: \( \text{true, eq}(\text{plus}(2, 2), 4), \text{eq}(2, x) \leftarrow \text{eq}(\text{succ}(2), \text{succ}(x)), \) and \( \text{geq}(x, 2) \rightarrow \exists x. \text{eq}(x, 2). \) \( \Box \)

### 2.3 Substitutions and unifiers

When defining operational principles of programming languages later on in this book, we will need certain notions concerning substitutions.

A substitution applied to a term or atomic formula replaces variables by terms.

**Definition 10** (substitution) A substitution \( \sigma \) is a function \( \sigma : X \to \mathcal{T}(F, X) \) with \( \sigma(x) \in \mathcal{T}(F, X)^s \) for all \( x \in X^s \).

We extend the function \( \sigma \) to \( \tilde{\sigma} : \mathcal{T}(F, X) \to \mathcal{T}(F, X) \), i.e. for application on terms by

- \( \tilde{\sigma}(x) = \sigma(x) \) for all variables \( x \in X \),
- \( \tilde{\sigma}(f(t_1, \ldots, t_n)) = f(\tilde{\sigma}(t_1), \ldots, \tilde{\sigma}(t_n)) \) for all terms \( f(t_1, \ldots, t_n) \).

Analogously, \( \sigma \) is extended for application on atomic formulae. In the following, we identify a substitution \( \sigma \) with its extension \( \tilde{\sigma} \) and write \( \sigma \) instead of \( \tilde{\sigma} \).

In this book, we deal with finite substitutions \( \sigma \) in the sense that for only finitely many variables \( x \) holds: \( \sigma(x) \neq x \). A substitution \( \sigma \) can, thus, be represented in the form \( \sigma = \{ x/\sigma(x) \mid \sigma(x) \neq x \} \), where we explicitly enumerate all its elements. We denote the identity substitution by \( \text{id} \).

The composition of substitutions describes the sequential application of substitutions on a term or formulae.

**Definition 11** (composition of substitutions) The composition of substitutions \( \sigma \) and \( \phi \) is defined by \( (\phi \circ \sigma)(e) = \phi(\sigma(e)) \) for all terms and atomic formulae \( e \).

**Example 5** Consider a set \( X \) of variables with \( \{ x, y, z \} \subseteq X \) and the signature \( \Sigma_\mathbb{N} \) from Example 1.

Let \( \sigma \) and \( \phi \) be substitutions with \( \sigma = \{ x/4, y/\text{plus}(3, z) \} \) and \( \phi = \{ z/1 \} \). The following holds:

- \( \sigma(\text{succ}(2)) = \text{succ}(\sigma(2)) = \text{succ}(2) \)
- \( \sigma(\text{succ}(x)) = \text{succ}(\sigma(x)) = \text{succ}(4) \)
- \( \sigma(\text{mul}(3, \text{plus}(x, \text{succ}(y)))) = \text{mul}(3, \text{plus}(4, \text{succ}(\text{plus}(3, z)))) \)
(ϕ ◦ σ)(\text{mul}(3, \text{plus}(x, \text{succ}(y)))) = φ(σ(\text{mul}(3, \text{plus}(x, \text{succ}(y)))))
= φ(\text{mul}(3, \text{plus}(4, \text{succ}(\text{plus}(3, z)))))
= \text{mul}(3, \text{plus}(4, \text{succ}(\text{plus}(3, 1))))

**Unifiers** are substitutions which allow one to identify certain terms or formulae.

**Definition 12** (unifier, most general unifier, mgu) Let s and t be terms or atoms. A substitution σ with σ(s) = σ(t) is a unifier of s and t. A unifier σ of s and t is a most general unifier (we write σ = mgu(s, t)) if for every unifier φ of s and t there exists a substitution ψ such that φ = ψ ◦ σ holds.  

For algorithms to compute most general unifiers we refer to [194, 106]. If two terms or atoms s and t are not unifiable, we write mgu(s, t) = ∅.1

**Example 6** Consider the signature \( \Sigma_\mathbb{N} \) and the set \{x, y, z\} ⊆ X. The terms \( s = \text{mul}(x, \text{succ}(z)) \) and \( t = \text{mul}(2, y) \) are unifiable with substitution \( σ = \{x/2, y/\text{succ}(z)\} \), i.e. σ is a unifier of s and t:

\[
σ(s) = \text{mul}(2, \text{succ}(z)) = σ(t)
\]

The substitution \( ϕ = \{x/2, y/\text{succ}(3), z/3\} \) is a unifier of s and t too:

\[
ϕ(s) = \text{mul}(2, \text{succ}(3)) = ϕ(t)
\]

The substitution σ is a most general unifier of s and t. For σ and ϕ there is a substitution ψ = \{z/3\} such that ϕ = ψ ◦ σ holds.  

Let the parallel composition of idempotent substitutions be defined as in [175]. We compute the parallel composition \((σ ↑ ϕ)\) of two idempotent substitutions σ and ϕ as follows:

\[
(σ ↑ ϕ) = \text{mgu}(f(x_1, \ldots, x_n, y_1, \ldots, y_m), f(σ(x_1), \ldots, σ(x_n), φ(y_1), \ldots, φ(y_m))),
\]

where \( x_i, i \in \{1, \ldots, n\}, \) and \( y_j, j \in \{1, \ldots, m\} \), are the domain variables of \( σ \) and \( ϕ \), resp.

**Example 7** Given \( \Sigma_\mathbb{N} \), a set X of variables with \{w, x, y, z\} ⊆ X, and the substitutions \( σ = \{x/0, y/z\}, ϕ = \{w/\text{succ}(x), y/0\}, \) and \( ψ = \{y/0, z/\text{succ}(0)\} \), we build their parallel compositions:

\[
(σ ↑ ϕ) = \text{mgu}(f(x, y, w, y), f(0, z, \text{succ}(x), 0))
= \{x/0, y/0, z/0, w/\text{succ}(0)\}
\]

\[
(σ ↑ ψ) = \text{mgu}(f(x, y, w, z), f(0, z, 0, \text{succ}(0))) = ∅
\]

\[
(ϕ ↑ ψ) = \text{mgu}(f(w, y, y, z), f(\text{succ}(x), 0, 0, \text{succ}(0)))
= \{w/\text{succ}(x), y/0, z/\text{succ}(0)\}
\]

---

1 Note that ∅ has a completely different meaning than the identity substitution id.
Multiparadigm Constraint Programming Languages
Hofstedt, P.
2011, XII, 180 p., Hardcover
ISBN: 978-3-642-17329-5