TECHNISCHE UNIVERSITÄT DRESDEN

Advanced Logics

Prof. Dr. Christel Baier

Technische Universität Dresden
Institute for Theoretical Computer Science

Summer Semester 2021

Preface

The purpose of this course is to study several variants of classical propositional and first-order logic. This includes several variants of first-order and second-order logic, monadic second-order logic and its connection to automata theory, modal logics and the lambda-calculus. We will concentrate here on aspects of mathematical logic that are relevant for Informatics, such as expressiveness, game-theoretic characterizations, deductive calculi, decision and transformation algorithms, and computational complexity.

The material of this course does not follow one particular text book. There is a wide range of excellent books that treat selected topics of the course, often in more detail than we will do. To mention a few of them we recommend [End02, EFT84, Pap94] (propositional and first-order logic), [Man96] (second-order logic), [Lib04] (finite model theory), and [HC96, BdRV01, FHMV95, Sti01] (modal logic).

References

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.

[EFT84] H.D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer-Verlag, 1984.

[End02] Herbert A. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2002.

[FHMV95] Ronald Fagin, Joseph Halpern, Yoram Moses, and Moshe Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[HC96] G.E. Hughes and M.J. Cresswell. *A New Introduction to Modal Logic*. Routledge Taylor & Francis Group, 1996.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[Man96] Maria Manzanno. *Extensions of First Order Logic*, volume 19 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1996.

[Pap94] Christos Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[Sti01] Colin Stirling. *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer, 2001.

1 First-order logic

We will start with a summary of the main features of first-order logic (FOL), also called predicate logic (Section 1.1) and important properties of FOL (Sections 1.2, 1.3 1.4 and 1.5). We then discuss the expressiveness and limitations of FOL (Section 1.6) and study some simple extensions of FOL (Section 1.7).

1.1 FOL in a nutshell

We first summarize some basic concepts of first-order logic (FOL) that will be central for the further sections. Most parts of the material of this section 1.1 should be known from other courses. It mainly serves to explain the notations used throughout the course. Whenever applicable, the same notations introduced here for FOL will also be used for other logics.

Syntax of FOL. Formulas of first-order logic (FOL-formulas) are built by the predicate and function symbols of a fixed vocabulary, variables and logical symbols for, e.g., negation, conjunction and universal quantification, and – for FOL with equality – the equality symbol $=$.

A *vocabulary* of first-order logic is a tuple $\mathit{Voc} = (\mathit{Pred}, \mathit{Func})$, where $\mathit{Pred} = (\mathit{Pred}_n)_{n \geqslant 1}$ and $\mathit{Func} = (\mathit{Func}_m)_{m \geqslant 0}$ are families of predicate and function symbols, respectively. In the literature sometimes the notion "signature" or "alphabet" is used instead of "vocabulary". The elements of Pred_n are n-ary *predicate symbols*, while the elements of Func_m are m-ary *function symbols*. Function symbols of arity 0 are called *constant symbols*. We often write *Const* for the set Func_0 of constant symbols.

The assumptions about the vocabulary that are made here are that (1) the sets Pred_n and Func_m are pairwise disjoint and (2) there is at least one predicate symbol, i.e., there is some $n \in \mathbb{N}$ such that Pred_n is nonempty. For FOL with equality, the latter requirement (2) can be dropped, as the equality symbol can be regarded as a predicate symbol with fixed semantics. We often identify the tuples *Pred* and *Func* with the sets $\bigcup_{n \geqslant 1} \mathit{Pred}_n$ and $\bigcup_{m \geqslant 0} \mathit{Func}_m$, respectively. Unless stated differently, *Pred* and *Func* are supposed to be recursively enumerable. This requirement is irrelevant at most places, and will only be crucial when talking about algorithmic aspects. Finiteness of a vocabulary means that the sets *Pred* and *Func* are finite. A vocabulary is called *relational* if it contains no function symbols of arity 1 or more, i.e., $\mathit{Func}_m = \varnothing$ for all $m \geqslant 1$. Relational vocabularies still might have constant symbols. By a *purely relational* vocabulary we mean a vocabulary $(\mathit{Pred}, \varnothing)$ which does not contain any function symbol.

For the variables we assume that they are chosen from some infinite countable set *Var*. When reasoning about computability questions, then *Var* is required to be recursively enumerable. The assumption that the set of symbols in the vocabularies and the variables are recursively enumerable ensures that the set of formulas is recursively enumerable. In particular, the set of formulas is countable.

In most cases, variables will be denoted with small letters at the end of the alphabet such as x, y, z or $x_1, x_2, \ldots$ Capital letters like P, Q, R will be used for predicate symbols and letters f, g, h for function symbols. We often use letters c, d for constant symbols.

$$\begin{array}{lcll}
t & ::= & x \mid f(t_1, \ldots, t_m) & \text{(terms)} \\[1em]
\phi & ::= & \mathit{true} \mid P(t_1, \ldots, t_n) \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \forall x.\phi \mid \underbrace{t_1 = t_2}_{\text{optional}} & \text{(formulas)}
\end{array}$$

Figure 1: Abstract syntax of terms and FOL-formulas

The *abstract syntax* of FOL-formulas (formulas for short) and terms over $\mathit{Voc} = (\mathit{Pred}, \mathit{Func})$ and *Var* is shown in Figure 1 (page 2) where $x \in \mathit{Var}$, $P \in \mathit{Pred}_n$, and $F \in \mathit{Func}_m$ (for some $n \geqslant 1$ and $m \geqslant 0$). Of course, we shall also need parentheses "(" and ")", which, for instance, are needed to distinguish $(\forall x.P(x,c)) \wedge Q(x)$ from $\forall x.(P(x,c)) \wedge Q(x))$. However, since the parentheses have no proper logical meaning, they are usually skipped in the abstract syntax.

The use of the equality symbol, i.e., atomic formulas of the form $t_1 = t_2$ where t_1 and t_2 are terms, is optional. At most places, it is irrelevant whether FOL with or without equality is considered. Unless stated differently, when speaking about FOL then we refer to FOL with equality over some fixed vocabulary.

Remark 1.1.1 (BNF-like notations for the abstract syntax). BNF-like notations as in Figure 1 for the abstract syntax of formulas will be used throughout the course. Let us briefly explain its meaning. The abstract syntax shown in Figure 1 is a shorthand notation to declare that the set $\mathit{Terms}(\mathit{Voc}, \mathit{Var})$ (whose elements are called of "terms over *Voc* and *Var*" or briefly "terms") is the smallest set where (T.1) and (T.2) hold:

(T.1) Each variable $x \in \mathit{Var}$ is a term.

(T.2) If $f \in \mathit{Func}_m$ and $t_1, \ldots, t_m$ are terms then so is $f(t_1, \ldots, t_m)$.

Note that by (T.2) with $m = 0$ we obtain that each constant symbol $c \in \mathit{Const}$ is a term.[1] Similarly, the set $\mathrm{FOL}(\mathit{Voc}, \mathit{Var})$ (whose elements are called "FOL-formulas over *Voc* and *Var*" or simply "formulas") is the smallest set such that (F.0), (F.1), (F.2) and (F.3) hold:

(F.0) *true* is a formula.

(F.1) If $P \in \mathit{Pred}_n$ and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is a formula.

(F.2) If ϕ, ϕ_1, ϕ_2 are formulas then so are $\neg\phi$ and $\phi_1 \wedge \phi_2$.

(F.3) If ϕ is a formula and $x \in \mathit{Var}$ then $\forall x.\phi$ is a formula.

For FOL with equality, formulas are built by (F.0)-(F.3) and the following (F.4):

(F.4) If t_1, t_2 are terms then $t_1 = t_2$ is a formula.

Formulas generated by (F.1) and (F.4) are called *atomic formulas*, also briefly called *atoms*. *Literals* are atomic formulas or their negations. ■

The abstract syntax shown in Figure 1 only uses the logical operators $\wedge$ (logical and, conjunction) and $\neg$ (negation) and universal quantification $\forall$. This choice, however, is arbitrary and we could have used any other basis (e.g., $\vee$, $\neg$ and $\exists$) as well. Other boolean connectors and existential quantification are derived in the standard way, see Figure 2.

To save parentheses, the standard precedence relation will be used where negation has highest priority, followed by universal or existential quantification, $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$ (in this order). E.g.:

$$\neg P(t) \wedge \forall x.Q(x) \vee R(x,y) \quad = \quad \big((\neg P(t)) \wedge (\forall x.Q(x))\big) \vee R(x,y)$$

We often make use of the associativity and commutativity of the semantics of conjunction $\wedge$ and disjunction $\vee$ and write, e.g., $\bigwedge\limits_{1 \leqslant i \leqslant n} \phi_i$ for $\phi_1 \wedge (\phi_2 \wedge (\ldots \wedge (\phi_{n-1} \wedge \phi_n) \ldots))$.

In the sequel, we often skip the explicit reference to the vocabulary and variable-set and simply speak about FOL-formulas or "the set of FOL-formulas", in which case we refer to FOL over some fixed vocabulary.

[1]For better readability we prefer the notation $f(t_1, \ldots, t_m)$ with paranthesis and comma between terms, although the precise notation should be $f t_1 t_2 \ldots t_m$. For the special case $m = 0$, i.e., if $f = c$ is a constant symbol, then we simply write c instead of $c()$.

In logic, a quantifier is an operator that specifies how many individuals in the domain of discourse satisfy an open formula. For instance, the universal quantifier {\displaystyle \forall } \forall in the first order formula x P (x) {\displaystyle \forall xP(x)} {\displaystyle \forall xP(x)} expresses that everything in the domain satisfies the property denoted by P {\displaystyle P} P. On the other hand, the existential quantifier {\displaystyle \exists } {\displaystyle \exists } in the formula x P (x) {\displaystyle \exists xP(x)} {\displaystyle \exists xP(x)} expresses that there is something in the domain which satisfies that property. A formula where a quantifier takes widest scope is called a quantified formula. A quantified formula must contain a bound variable and a subformula specifying a property of the referent of that variable.

The mostly commonly used quantifiers are {\displaystyle \forall } \forall and {\displaystyle \exists } \exists . These quantifiers are standardly defined as duals and are thus interdefinable using negation. They can also be used to define more complex quantifiers, as in the formula ¬ x P (x) {\displaystyle \neg \exists xP(x)} {\displaystyle \neg \exists xP(x)} which expresses that nothing has the property P {\displaystyle P} P. Other quantifiers are only definable within second order logic or higher order logics. Quantifiers have been generalized beginning with the work of Mostowski and Lindström.

First order quantifiers approximate the meanings of some natural language quantifiers such as "some" and "all". However, many natural language quantifiers can only be analyzed in terms of generalized quantifiers.

$$\begin{array}{lcll}
\phi_1 \vee \phi_2 & \stackrel{\text{def}}{=} & \neg(\neg\phi_1 \wedge \neg\phi_2) & \text{(logical or, disjunction)} \\
\phi_1 \rightarrow \phi_2 & \stackrel{\text{def}}{=} & \neg\phi_1 \vee \phi_2 & \text{(implication)} \\
\phi_1 \leftrightarrow \phi_2 & \stackrel{\text{def}}{=} & (\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2) & \text{(equivalence)} \\
\phi_1 \oplus \phi_2 & \stackrel{\text{def}}{=} & (\phi_1 \wedge \neg\phi_2) \vee (\neg\phi_1 \wedge \phi_2) & \text{(xor, parity)} \\
\exists x.\phi & \stackrel{\text{def}}{=} & \neg\forall x.\neg\phi & \text{(existential quantification)} \\
t_1 \neq t_2 & \stackrel{\text{def}}{=} & \neg(t_1 = t_2) & \\
\mathit{false} & \stackrel{\text{def}}{=} & \neg \mathit{true} & \\
 & \vdots & &
\end{array}$$

Figure 2: Derived logical operators

Free variables. An occurrence of a variable x in a formula is said to be bounded if it is in the scope of a quantifier. Otherwise it is called free. E.g., the first occurrence of x in the formula $P(x) \wedge \forall x.Q(x, f(c))$ is free, while the second is bounded. The set $\mathit{Free}(\phi)$ of variables that have free occurrences in ϕ is given by:

$$\begin{array}{lcl}
\mathit{Free}(\mathit{true}) & \stackrel{\text{def}}{=} & \varnothing \\
\mathit{Free}(\,P(t_1,\ldots,t_n)\,) & \stackrel{\text{def}}{=} & \text{set of variables that appear in } t_1,\ldots,t_n \\
\mathit{Free}(\neg\phi) & \stackrel{\text{def}}{=} & \mathit{Free}(\phi) \\
\mathit{Free}(\phi_1 \wedge \phi_2) & \stackrel{\text{def}}{=} & \mathit{Free}(\phi_1) \cup \mathit{Free}(\phi_2) \\
\mathit{Free}(\forall x.\phi) & \stackrel{\text{def}}{=} & \mathit{Free}(\phi) \setminus \{x\}
\end{array}$$

For FOL with equality, $\mathit{Free}(t_1 = t_2)$ is the set of the variables that appear in t_1 or t_2. Variable x is said to be free in ϕ if $x \in \mathit{Free}(\phi)$, i.e., if x has at least one free occurrence in ϕ.

It is sometimes awkward to deal with formulas in which variables have both free and bounded occurrences. Using the concept of *bounded renaming* (i.e., replacing bounded variables with fresh variables), each formula ϕ can be transformed into an equivalent formula ϕ' where no variable has both free and bounded occurrences in ϕ'. E.g., $P(x) \wedge \forall x.\big(Q(x, f(c)) \vee \neg R(x)\big)$ can be replaced with $P(x) \wedge \forall y.\big(Q(y, f(c)) \vee \neg R(y)\big)$.

Sentences, closed formulas, generalizations. Formula ϕ is called *closed*, or a *sentence*, if $\mathit{Free}(\phi) = \varnothing$. If ψ is a formula then all formulas of the form

$$\forall x_1 \ldots \forall x_n.\psi,$$

where $x_1, \ldots, x_n$ are variables, are called *generalizations* of ψ. Thus, if $\mathit{Free}(\psi) \subseteq \{x_1, \ldots, x_n\}$ then the generalization $\forall x_1 \ldots \forall x_n.\psi$ is a sentence.

Substitution. If $x_1, \ldots, x_n$ are pairwise distinct variables and $t_1, \ldots, t_n$ are terms then

$$\phi[x_1/t_1, \ldots, x_n/t_n]$$

denotes the formula that results from ϕ by replacing simultaneously all free occurrences of x_i in ϕ with t_i. Sometimes $[x_1/t_1, \ldots, x_n/t_n]$ is called a *substitution*. At several places we use the notation

$$\phi(x_1,\ldots,x_n)$$

to indicate that $x_1,\ldots,x_n$ are pairwise distinct variables with $\mathit{Free}(\phi) \subseteq \{x_1,\ldots,x_n\}$. In this case, $\phi(t_1,\ldots,t_n)$ stands short for $\phi[x_1/t_1,\ldots,x_n/t_n]$.

Tuple notations. To simplify notations, we shall also use tuples of terms and variables and write, e.g., $\phi[\overline{x}/\overline{t}]$ or $\phi(\overline{t})$ as a shortform notation for $\phi[x_1/t_1,\ldots,x_n/t_n]$, where $\overline{t} = (t_1,\ldots,t_n)$ and $\overline{x} = (x_1,\ldots,x_n)$. Similarly, $\exists\overline{x}.\phi$ and $\forall\overline{x}.\phi$ stands short for $\exists x_1 \ldots \exists x_n.\phi$ and $\forall x_1 \ldots \forall x_n.\phi$, respectively. Here and in the sequel, we skip the dot between consecutive quantifiers and write, e.g., $\forall x \forall y.\psi$ rather than $\forall x.\forall y.\psi$.

Subformulas. For formula ϕ, the set $\mathit{subf}(\phi)$ of *subformulas* of ϕ is given by:

$$\begin{array}{lcll}
\mathit{subf}(\phi) & \overset{\text{def}}{=} & \{\phi\} & \text{if } \phi \text{ is } \mathit{true} \text{ or an atomic formula} \\
\mathit{subf}(\neg\phi) & \overset{\text{def}}{=} & \{\neg\phi\} \cup \mathit{subf}(\phi), & \\
\mathit{subf}(\phi_1 \wedge \phi_2) & \overset{\text{def}}{=} & \{\phi_1 \wedge \phi_2\} \cup \mathit{subf}(\phi_1) \cup \mathit{subf}(\phi_2), & \\
\mathit{subf}(\forall x.\phi) & \overset{\text{def}}{=} & \{\forall x.\phi\} \cup \mathit{subf}(\phi). &
\end{array}$$

Length and word-length of formulas. The *length* of a formula ϕ, denoted $|\phi|$, is defined as the number of operators in ϕ:

$$\begin{array}{rcll}
|\phi| & \overset{\text{def}}{=} & 0 & \text{if } \phi \text{ is } \mathit{true} \text{ or an atomic formula} \\
|\neg\phi| & \overset{\text{def}}{=} & |\phi| + 1 & \\
|\phi_1 \wedge \phi_2| & \overset{\text{def}}{=} & |\phi_1| + |\phi_2| + 1 & \\
|\forall x.\phi| & \overset{\text{def}}{=} & |\phi| + 1 &
\end{array}$$

This definition of the length of a formula is adequate for inductive reasoning. One often speaks about *structural induction* to denote an inductive proof or definition that is provided by induction on the length of derivations of formulas from the grammar shown in Figure 1. That is, the basis of induction treats the formula *true* and atomic formulas. In the step of induction, one assumes that for a given formula ϕ, the proper subformulas have already been treated (induction hypothesis) and provides the proof or definition for ϕ. This roughly agrees with an ordinary induction on the length of formulas. For instance, the above definitions of $\mathit{Free}(\phi)$ and $|\phi|$ are by structural induction.

The length $|\phi|$ relates to the number of subformulas by:

$$|\mathit{subf}(\phi)| \leqslant 2|\phi| + 1 = \mathcal{O}(|\phi|)$$

The proof of this relation between the length and the number of subformulas is by structural induction. In the basis of induction we regard a FOL-formula ϕ of length 0. That is, ϕ is *true* or an atomic formula. But then, $|\mathit{subf}(\phi)| = 1 = 2 \cdot 0 + 1 = 2|\phi| + 1$. In the step of induction, we suppose that $|\phi| \geqslant 1$. That is, the outermost logical operator of ϕ is either a conjunction $\wedge$, a negation $\neg$ or a universal quantifier. For $\phi = \forall x.\psi$ or $\phi = \neg\psi$, we have $|\mathit{subf}(\phi)| = |\mathit{subf}(\psi)| + 1$ and $|\phi| = |\psi| + 1$. The induction hypothesis (IH) yields $|\mathit{subf}(\psi)| \leqslant 2|\psi| + 1$. Hence, we get:

$$|\mathit{subf}(\phi)| \leqslant |\mathit{subf}(\psi)| + 1 \leqslant 2|\psi| + 2 = 2|\phi| < 2|\phi| + 1$$

For $\phi = \phi_1 \wedge \phi_2$ we have:

$$\begin{aligned}
|\mathit{subf}(\phi)| &\leqslant |\mathit{subf}(\phi_1)| + |\mathit{subf}(\phi_2)| + 1 \\
&\overset{\text{IH}}{\leqslant} (2|\phi_1|+1) + (2|\phi_2|+1) + 1 \\
&= 2(|\phi_1|+|\phi_2|+1) + 1 \\
&= 2|\phi|+1
\end{aligned}$$

The length $|\phi|$ does not incorporate the length of the terms that appear in ϕ. Another measure of the formula-length is the *word-length* $||\phi||$ which counts the total number of occurrences of symbols (variables, function and predicate symbols, the logical symbols $\neg$ and $\wedge$ and quantifiers $\forall x$) that appear in ϕ. This measure $||\cdot||$ is appropriate for complexity theoretic considerations of algorithms that take as input one or more FOL-formulas. It is defined for terms and formulas as shown in Figure 3 in page 6. Since we are only interested in the asymptotic word-length

$$\begin{array}{rcll}
||x|| & \overset{\text{def}}{=} & 1 & \text{if } x \text{ is a variable} \\
||c|| & \overset{\text{def}}{=} & 1 & \text{if } c \text{ is a constant symbol} \\
||f(t_1,\ldots,t_n)|| & \overset{\text{def}}{=} & 1 + ||t_1|| + \ldots + ||t_n|| & \\
\hline
||\mathit{true}|| & \overset{\text{def}}{=} & 1 & \\
||P(t_1,\ldots,t_n)|| & \overset{\text{def}}{=} & 1 + ||t_1|| + \ldots + ||t_n|| & \\
||\neg\phi|| & \overset{\text{def}}{=} & ||\phi|| + 1 & \\
||\phi_1 \wedge \phi_2|| & \overset{\text{def}}{=} & ||\phi_1|| + ||\phi_2|| + 1 & \\
||\forall x.\phi|| & \overset{\text{def}}{=} & ||\phi|| + 1 &
\end{array}$$

Figure 3: Word-length of terms and FOL-formulas

of formulas, the above definition of the word-length ignores the parenthesis. Furthermore, for the asymptotic word-length it is irrelevant whether the length of a quantification $\forall x$ is defined to be 1 (as we did in Figure 3) or 2 (one for $\forall$ and one for x) and whether the derived operators $\vee$, $\rightarrow$ and $\exists$ are treated by their syntactic definitions (e.g., $||\phi_1 \vee \phi_2|| = ||\neg(\neg\phi_1 \wedge \neg\phi_2)|| = ||\phi_1|| + ||\phi_2|| + 4$) or as basic operators where any occurrence of $\vee$, $\rightarrow$ and $\exists$ causes one word-length unit. The latter means to define, e.g., $||\phi_1 \vee \phi_2||$ as $||\phi_1|| + ||\phi_2|| + 1$.

The same holds for asymptotic considerations on the length $|\phi|$ for formulas. However, one should be careful with derived operators other than $\vee$, $\rightarrow$ or $\exists$. For instance, the equivalence operator $\leftrightarrow$ and the parity operator $\oplus$, can cause a duplication of the (word-)lengths.

For finite relational vocabularies and asymptotic considerations, it is irrelevant whether one deals with the length $|\phi|$ or word-length $||\phi||$. Recall that in relational vocabularies, terms are just variables or constant symbols. Thus, $||\phi|| = \Theta(|\phi|)$ for FOL-formulas ϕ over finite relational vocabularies.

$$\begin{array}{lll}
\mathcal{I} \models \mathit{true} & & \\
\mathcal{I} \models P(t_1,\ldots,t_n) & \text{iff} & (t_1^{\mathcal{I}},\ldots,t_n^{\mathcal{I}}) \in P^{\mathcal{A}} \\
\mathcal{I} \models \phi_1 \wedge \phi_2 & \text{iff} & \mathcal{I} \models \phi_1 \text{ and } \mathcal{I} \models \phi_2 \\
\mathcal{I} \models \neg\phi & \text{iff} & \mathcal{I} \not\models \phi \\
\mathcal{I} \models \forall x.\phi & \text{iff} & \mathcal{I}[x := a] \models \phi \text{ for all } a \in A \\
\mathcal{I} \models (t_1 = t_2) & \text{iff} & t_1^{\mathcal{I}} = t_2^{\mathcal{I}}
\end{array}$$

Figure 4: Satisfaction relation for FOL-formulas

Semantics of FOL. FOL-formulas are interpreted over pairs $(\mathcal{A},\mathcal{V})$ consisting of a structure that provides the meaning of the predicate and function symbols (by assigning predicates and functions of corresponding arity to them) and a variable valuation $\mathcal{V}$ that interpretes the variables by concrete elements. The variable valuation is only relevant for the free variables. Thus, the truth value of a sentence just depends on the interpretation of the elements in the vocabulary (i.e., the predicate and function symbols).

Structure. A *structure* for a vocabulary $\mathit{Voc} = (\mathit{Pred},\mathit{Func})$ is a tuple

$$\mathcal{A} \;=\; \big(A, (P^{\mathcal{A}})_{P\in \mathit{Pred}}, (f^{\mathcal{A}})_{f\in \mathit{Func}}\big)$$

consisting of a nonempty set A, called *domain* (or *universe*), and denoted by $\mathit{Dom}^{\mathcal{A}}$, and tuples that provide an interpretation of the predicate and function symbols by predicates and functions of the corresponding arity. More precisely, $P^{\mathcal{A}} \subseteq A^n$ if $P \in \mathit{Pred}_n$ and $f^{\mathcal{A}} : A^m \to A$ if f is an m-ary function symbol. For the special case of a 0-ary function symbol c (i.e., a constant symbol), $c^{\mathcal{A}}$ can be viewed as an element of A. Occasionally, we skip the superscript $\mathcal{A}$ and simply write P or f rather than $P^{\mathcal{A}}$ or $f^{\mathcal{A}}$, respectively. $\mathcal{A}$ is called *finite* if $A = \mathit{Dom}^{\mathcal{A}}$ is finite. Similarly, $\mathcal{A}$ is said to be countable (uncountable, infinite) if its domain A is.

Interpretations. Let Voc be a vocabulary and Var a set of variables. An *interpretation* for $(\mathit{Voc},\mathit{Var})$ is a pair $\mathcal{I} = (\mathcal{A},\mathcal{V})$ consisting of a structure $\mathcal{A}$ for Voc and a variable valuation $\mathcal{V} : \mathit{Var} \to A$ where $A = \mathit{Dom}^{\mathcal{A}}$. We refer to the interpretations $(\mathcal{A},\mathcal{V})$ as $\mathcal{A}$-interpretations. If $\mathcal{I}$ is an $\mathcal{A}$-interpretation and A the domain of $\mathcal{A}$ and t is a term then $t^{\mathcal{I}} \in A$ denotes its meaning under $\mathcal{I}$, i.e.,

- $x^{\mathcal{I}} \overset{\text{def}}{=} \mathcal{V}(x)$ for each variable x,
- $f(t_1,\ldots,t_m)^{\mathcal{I}} \overset{\text{def}}{=} f^{\mathcal{A}}(t_1^{\mathcal{I}},\ldots,t_m^{\mathcal{I}})$ for each m-ary constant symbol f.

With $m = 0$ we get $c^{\mathcal{I}} = c^{\mathcal{A}}$ for each constant symbol c.

Satisfaction relation $\models$. The satisfaction relation $\models$ for interpretations $\mathcal{I}$ and formulas ϕ is defined by structural induction as shown in Figure 4, where $\mathcal{I}[x := a]$ agrees with $\mathcal{I}$, except that the variable valuation assigns a to x. Formally, if $\mathcal{I} = (\mathcal{A},\mathcal{V})$ then $\mathcal{I}[x := a] = (\mathcal{A},\mathcal{V}[x := a])$ where

$$\mathcal{V}[x := a](y) = \begin{cases} \mathcal{V}(y) & : \text{ if } y \neq x \\ a & : \text{ if } x = y \end{cases}$$

Similarly, if $x_1,\ldots,x_n$ are pairwise distinct variables and $a_1,\ldots,a_n \in A$ then $\mathcal{I}[x_1 := a_1,\ldots,x_n := a_n]$ denotes the interpretation $(\mathcal{A},\mathcal{V}[x_1 := a_1,\ldots,x_n := a_n])$ where

$$\mathcal{V}[x_1 := a_1,\ldots,x_n := a_n](y) \;=\; \begin{cases} \mathcal{V}(y) & : \text{ if } y \notin \{x_1,\ldots,x_n\} \\ a_i & : \text{ if } y = x_i \end{cases}$$

If $\mathcal{I} \models \phi$ then ϕ is said to be *true* (or to *hold* or to be *satisfied*) in $\mathcal{I}$. Clearly, the truth of ϕ in $\mathcal{I} = (\mathcal{A},\mathcal{V})$ only depends on the values $\mathcal{V}(x)$ for the variables x that appear free in ϕ. Thus, for all variable valuations $\mathcal{V}$ and $\mathcal{V}'$ that differ at most in the variables $x \in \mathit{Var} \setminus \mathit{Free}(\phi)$ we have: $(\mathcal{A},\mathcal{V}) \models \phi$ if and only if $(\mathcal{A},\mathcal{V}') \models \phi$. In particular, the truth value of sentences under an interpretation just depends on the structure. We often use sloppy notations such as

$$(\mathcal{A},a_1,\ldots,a_n) \models \phi(x_1,\ldots,x_n)$$

where $(\mathcal{A},a_1,\ldots,a_n)$ stands for some $\mathcal{A}$-interpretation $(\mathcal{A},\mathcal{V})$ that interpretes variable x_i by the element $a_i \in A$ (i.e., $\mathcal{V}(x_i) = a_i$ for $i = 1,\ldots,n$). Recall that the notation $\phi(x_1,\ldots,x_n)$ indicates that $x_1,\ldots,x_n$ are pairwise distinct variables and $\mathit{Free}(\phi) \subseteq \{x_1,\ldots,x_n\}$.

Models. An interpretation $\mathcal{I}$ is called a *model* for ϕ if $\mathcal{I} \models \phi$. An interpretation $\mathcal{I}$ is called a model for a formula-set $\mathfrak{F}$, denoted $\mathcal{I} \models \mathfrak{F}$, if $\mathcal{I}$ is a model for all formulas $\phi \in \mathfrak{F}$. That is:

$$\mathcal{I} \models \mathfrak{F} \quad \text{iff} \quad \mathcal{I} \models \phi \text{ for all } \phi \in \mathfrak{F}$$

A structure $\mathcal{A}$ is called a model for a formula ϕ if ϕ is true in $\mathcal{A}$, no matter how the variables are interpreted. Formally, a structure $\mathcal{A}$ is called a model for formula ϕ, denoted $\mathcal{A} \models \phi$, iff $(\mathcal{A},\mathcal{V}) \models \phi$ for all variable valuations $\mathcal{V} : \mathit{Var} \to A$ (where $A = \mathit{Dom}^{\mathcal{A}}$). Thus, if $\mathit{Free}(\phi) = \{x_1,\ldots,x_n\}$ then

$$\mathcal{A} \models \phi \quad \text{iff} \quad \mathcal{A} \models \forall x_1 \ldots \forall x_n.\phi.$$

Similarly, a structure $\mathcal{A}$ is said to be a model for a formula-set $\mathfrak{F}$, denoted $\mathcal{A} \models \mathfrak{F}$, if all $\mathcal{A}$-interpretations $(\mathcal{A},\mathcal{V})$ are models for $\mathfrak{F}$. That is:

$$\mathcal{A} \models \mathfrak{F} \quad \text{iff} \quad \mathcal{A} \models \phi \text{ for all formulas } \phi \in \mathfrak{F}$$

Substitution lemma. The following observation is often useful to evaluate FOL-formulas of the form $\phi[x_1/t_1,\ldots,x_n/t_n]$. We say that a variable x *can be replaced with a term* t in a formula ϕ iff there is no variable y that appears in t such that y has bounded occurrences in ϕ by means of subformulas $\forall y.\psi$ of ϕ where $x \in \mathit{Free}(\psi)$. This condition applies to any ground term, i.e., term without variables. For another example, variable x can be replaced in $\phi = \exists y.P(x,y)$ with each term t that does not contain variable y. However, x cannot be replaced with $t = y$ in $\exists y.P(x,y)$. The intuitive explanation of this notation is that the replacement of x with y in $\exists y.P(x,y)$ invokes a new (typically undesired) binding as we have $(\exists y.P(x,y))[x/y] = \exists y.P(y,y)$.

The statement of the substitution lemma is as follows. Let $\mathcal{I} = (\mathcal{A},\mathcal{V})$ be an interpretation and let $x_1,\ldots,x_n$ be pairwise distinct variables and $t_1,\ldots,t_n$ terms such that x_i can be replaced in ϕ with t_i for $1 \leqslant i \leqslant n$. Then:

$$\mathcal{I} \models \phi[x_1/t_1,\ldots,x_n/t_n] \quad \text{iff} \quad \mathcal{I}\big[x_1 := t_1^{\mathcal{I}},\ldots,x_n := t_n^{\mathcal{I}}\,\big] \models \phi$$

Validity. Formula ϕ is called *valid* (or a *tautology*) if $\mathcal{I} \models \phi$ for all interpretations $\mathcal{I}$, i.e., if all structures are models for ϕ. We sometimes write $\Vdash \phi$ to denote that ϕ is valid. For example, if variable x can be replaced with term t in formula ϕ then

$$\Vdash \forall x.\phi \rightarrow \phi[x/t]$$

This is a consequence of the substitution lemma, since for each interpretation $\mathcal{I} = (\mathcal{A}, \mathcal{V})$ with domain A we have:

$$\begin{array}{lll} & \mathcal{I} \models \forall x.\phi & \\ \Longrightarrow & \mathcal{I}[x := a] \models \phi & \text{for all } a \in A \\ \Longrightarrow & \mathcal{I}[x := t^{\mathcal{I}}] \models \phi & \\ \Longrightarrow & \mathcal{I} \models \phi[x/t] & \text{by the substitution lemma} \end{array}$$

Propositional tautology. A FOL-formula ϕ is said to be a *propositional tautology* if ϕ arises from a valid propositional formula φ by replacing uniformly the atomic propositions (boolean variables) in φ with FOL-formulas. For example, if q and p are atomic propositions then $\varphi = (q \wedge p) \rightarrow p$ is a valid propositional formula. We now may replace the atomic propositions q and p with arbitrary FOL-formulas ϕ and ψ to obtain the propositional tautology $\varphi[q/\phi, p/\psi] = (\phi \wedge \psi) \rightarrow \psi$. Thus, e.g., the formula $(\,(\forall x.P(x)) \wedge \exists y.R(y)\,) \rightarrow \exists y.\,R(y)$ is a propositional tautology.

Satisfiability. Formula ϕ is called *satisfiable* if $\mathcal{I} \models \phi$ for some interpretation $\mathcal{I}$. Similarly, a formula-set $\mathfrak{F}$ is called *satisfiable* if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \phi$ for all $\phi \in \mathfrak{F}$. Formula-set $\mathfrak{F}$ is called

- *finitely satisfiable* if each finite subset of $\mathfrak{F}$ is satisfiable,
- *satisfiable over structure* $\mathcal{A}$ if there exists a valuation $\mathcal{V}$ such that $(\mathcal{A}, \mathcal{V}) \models \mathfrak{F}$,
- *true in structure* $\mathcal{A}$ if $\mathcal{A} \models \phi$ for all formulas $\phi \in \mathfrak{F}$.

The notions “satisfiability” and “truth in some structure” agree for sets of sentences, while they differ for formula-sets $\mathfrak{F}$ which contain formulas with free variables. For example, $\mathfrak{F} = \{P(x), \neg P(y)\}$ is satisfiable (consider the structure with domain $A = \{0,1\}$, the predicate $P^{\mathcal{A}} = \{0\}$ and variable valuation $\mathcal{V}$ with $\mathcal{V}(x) = 0$, $\mathcal{V}(y) = 1$), but there is no structure $\mathcal{A}$ such that $\mathcal{A} \models \mathfrak{F}$. Thus, truth in some structure implies satisfiability, while the reverse implication only holds for formula-sets without free variables.

Equivalence $\equiv$. Formulas ϕ and ψ are called *equivalent*, denoted $\phi \equiv \psi$, if ϕ and ψ yield the same truth value for all interpretations. That is:

$$\phi \equiv \psi \quad \text{iff} \quad \left\{ \begin{array}{l} \text{for all interpretations } \mathcal{I}\text{:} \\ \quad \mathcal{I} \models \phi \Longleftrightarrow \mathcal{I} \models \psi \end{array} \right.$$

Logical consequences, consequence relation $\Vdash$, logical closure. If $\mathfrak{F}$ is a formula-set and ϕ a formula then ϕ is called a (semantical, logical) *consequence* of $\mathfrak{F}$, denoted $\mathfrak{F} \Vdash \phi$, if every model for $\mathfrak{F}$ is also a model for ϕ, i.e.:

$$\mathfrak{F} \Vdash \phi \quad \text{iff} \quad \begin{cases} \text{for all interpretations } \mathfrak{I} = (\mathcal{A}, \mathcal{V}): \\ \qquad \text{if } \mathfrak{I} \models \mathfrak{F} \text{ then } \mathfrak{I} \models \phi \end{cases}$$

If $\mathfrak{F} \Vdash \phi$ then $\mathfrak{F}$ is said to *imply* ϕ. For formulas ϕ and ψ, we write $\psi \Vdash \phi$ iff $\{\psi\} \Vdash \phi$. If $\mathfrak{F}$ is a formula-set then the set

$$Cl(\mathfrak{F}) \stackrel{\text{def}}{=} \{\phi : \mathfrak{F} \Vdash \phi\}$$

is called the *logical closure* of $\mathfrak{F}$. Obviously, we have:

- $\mathfrak{F} \Vdash \phi$ iff $\mathfrak{F} \cup \{\neg\phi\}$ is not satisfiable
- $\phi \equiv \psi$ iff $\phi \Vdash \psi$ and $\psi \Vdash \phi$
- ϕ is valid iff $\neg\phi$ is not satisfiable iff *true* $\Vdash \phi$ iff $\psi \Vdash \phi$ for all formulas ψ
- ϕ is not satisfiable iff $\neg\phi$ is valid iff $\phi \Vdash$ *false* iff $\phi \Vdash \psi$ for all formulas ψ.

Here is a proof of the first statement. We show that $\mathfrak{F} \nVdash \phi$ if and only if $\mathfrak{F} \cup \{\neg\phi\}$ is satisfiable. For the implication "$\Longleftarrow$" we suppose that $\mathfrak{I}$ is a model for $\mathfrak{F} \cup \{\neg\phi\}$. Then, $\mathfrak{I}$ is a model for $\mathfrak{F}$ such that $\mathfrak{I} \not\models \phi$. Hence, $\mathfrak{F} \nVdash \phi$. For the implication "$\Longrightarrow$", we suppose that $\mathfrak{F} \nVdash \phi$. Then, there is a model $\mathfrak{I}$ for $\mathfrak{F}$ such that $\mathfrak{I} \not\models \phi$. But then, $\mathfrak{I} \models \neg\phi$ and $\mathfrak{I}$ is a model for $\mathfrak{F} \cup \{\neg\phi\}$.