Advanced CS 1: Formal Languages and Logic
Part 2: Logic

Florian Rabe

2008-2013

Contents

Standard textbooks for logic are [Gal86], [Smu95], and [End72] (look for the respective latest edition). [And86] is an interesting textbook that emphasizes proof theory over model theory.

Chapter 1

Syntax

1.1 Syntax of Propositional Logic

Definition 1.1 (PL-signatures)**.** A PL-signature is an alphabet, i.e., a set Σ of symbols.

Definition 1.2 (PL formulas)**.** The PL-grammar for the signature Σ is:

`form`	::=	*true*	*truth*
	|	*false*	*falsity*
	|	`form` $\wedge$ `form`	*conjunction*
	|	`form` $\vee$ `form`	*disjunction*
	|	`form` $\to$ `form`	*implication*
	|	$\neg$`form`	*negation*
	|	p where $p \in \Sigma$	*boolean variables*

The set of words over this grammar is the set $\mathbf{Sen}^{PL}(\Sigma)$ of Σ-*formulas* (also called Σ-*propositions* or Σ-*sentences*).

This grammar is of course ambiguous. If we actually want to parse words, we have to make it unambiguous by adding brackets around conjunctions, disjunctions, and implications.

Remark 1.3. It is a specialty of propositional logic that all formulas are well-formed, i.e., the well-formed PL-formulas over Σ are a context-free language. Normally, the languages are context-sensitive.

1.2 Syntax of First-Order Logic

Definition 1.4 (FOL-Signatures)**.** A FOL-signature is a triple (Σ_p, Σ_f, ar) where Σ_p and Σ_f are disjoint sets of symbols and $ar : \Sigma_p \cup \Sigma_f \to \mathbb{N}$ is a mapping.

The symbols in Σ_p and Σ_f are called *predicate symbols* and *function symbols*, respectively. If $ar(s) = 0, 1, 2, 3, 4, \ldots$, we say that s is a *nullary*, *unary*, *binary*, *ternary*, 4-ary, ..., symbol.

Definition 1.5 (FOL-Grammar)**.** The FOL-grammar for the signature $\Sigma = (\Sigma_p, \Sigma_f, ar)$ is given by the productions in Fig. 1.1 with start symbol `form`.

The language of this grammar is denoted by $Form^{FOL}(\Sigma)$.

Words produced from `term` are called *terms*. Words produced from `form` are called *formulas* (or propositions or sentences).

Words produced from `var` are variable names. These are elements of some countable set that is usually left implicit. For example, we can put

$$L(\texttt{var}) = \{x_n, y_n, z_n \mid n \in \mathbb{N}\}.$$

This grammar is also ambiguous. To make it unambiguous, we have to add brackets around all complex formulas.

`form`	::=	*true*	*truth*
	|	*false*	*falsity*
	|	`form` $\wedge$ `form`	*conjunction*
	|	`form` $\vee$ `form`	*disjunction*
	|	`form` $\rightarrow$ `form`	*implication*
	|	$\neg$`form`	*negation*
	|	$\forall$`var form`	*universal quantification*
	|	$\exists$`var form`	*existential quantification*
	|	$p(\underbrace{\texttt{term}, \ldots, \texttt{term}}_{n})$ where $p \in \Sigma_p$ with $ar(p) = n$	*atomic formulas* or *predicates*
`term`	::=	$f(\underbrace{\texttt{term}, \ldots, \texttt{term}}_{n})$ where $f \in \Sigma_f$ with $ar(f) = n$	
	|	`var`	
`var`	::=	some symbol	*variables*

Figure 1.1: FOL Grammar

Remark 1.6. The intuition behind the arity is that it gives the number of arguments that a function or predicate symbol takes.

Arity 0 is permitted: Nullary function symbols are also called *constant* symbols, nullary predicate symbols are the *boolean variables* from propositional logic.

Remark 1.7. The intuition behind the non-terminals `term` and `form` is that terms represent mathematical objects and formulas represent properties of mathematical objects.

It is easy to see that the FOL syntax is an extension of the PL-syntax.

Note that formulas and terms are not mixed: In every branch of the syntax tree of a well-formed formula, all `form`-nodes occur above all `term`-nodes.

Equality is needed so often that we define FOLEQ. It is like FOL but with a fixed binary predicate symbol $\doteq$ that is always present.

Definition 1.8 (FOL with equality)**.** FOLEQ is defined like FOL but with one production added to the grammar: `form` ::= `term` $\doteq$ `term`.

Remark 1.9 (Equality). It is crucial to understand the difference between $=$ and $\doteq$.

$\doteq$ is a symbol of the alphabet, which happens to look very similar to the symbol $=$. But at this point $\doteq$ has no meaning whatsoever. It just takes two term and turns them into a formula.

$=$ on the other hand, has a meaning: It denotes equality of mathematical objects. In particular, we can use $=$ to express the equality of words over our alphabets, e.g., the equality of terms and formulas. For example, we can say the following: "We assume $F = s \doteq t$." This sentence expresses that F is a word (in this case a formula) over the alphabet and F is the formula $s \doteq t$.

Example 1.10 (Set Theory). Axiomatic set theory was introduced around the 1920s by Zermelo and Fraenkel. It is the base of all modern mathematics. In modern language and notation, a very simple variant of axiomatic set theory is based on the following FOLEQ-signature:

- a binary predicate symbol $\in$, and we usually write $s \in t$ instead of $\in (s, t)$
- a nullary function symbol $\varnothing$,
- a unary function symbol Pow,
- a unary function symbol $\bigcup$,
- a binary function symbol $unorderedpair$, and we usually write $\{s, t\}$ instead of $unorderedpair(s, t)$.

Example formulas for this signature are

- $\forall x\ \neg x \in \varnothing$,
- $\forall x\ \forall y\ x \in \{x, y\}$,
- $\forall x\ \forall y\ \{x, y\} \doteq \{y, x\}$,
- $\forall x\ \varnothing \doteq \{x, x\}$,
- $\forall x\ \forall y\ x \in y$.

Note the difference between the first three and the last two formulas: The first three feel "correct" or "true", whereas the last two feel "false". However, at this point, no formula has a meaning. They are all just words over our grammar.

This signature looks like it has only very few and very boring terms, for example $\varnothing$, $Pow(\varnothing)$, $\{\varnothing, \varnothing\}$, and so on. And the formulas are even more boring because there is only one predicate symbol. However, together with some abbreviations, this is already enough to talk about most of mathematics.

For example, mathematics usually uses the following abbreviations for terms and formulas:

- If s and t are terms, then the term $\{s, \{s, t\}\}$ is abbreviated as (s, t).
- If s and t are terms, then the term $\bigcup\{s, t\}$ is abbreviated as $s \cup t$ (or $(s \cup t)$ if there are ambiguities otherwise).
- If s is a term, then the term $\{s, s\}$ is abbreviated as $\{s\}$.
- If s and t are terms (which do not use the variable x), then the formula $\forall x\ (x \in s \to x \in t)$ is abbreviated as $s \subseteq t$.

Then we can give even more abbreviations:

- We write 0 instead of $\varnothing$.
- We write 1 instead of $\{\varnothing\}$.
- We write 2 instead of $\{\varnothing, \{\varnothing\}\}$.
- We write 3 instead of $2 \cup \{2\}$.
- and so on

Example 1.11 (Monoids and Groups). Monoids and groups (as well as rings, fields, etc.) can also be expressed as FOLEQ signatures. The signature for monoids has:

- a binary function symbol $\circ$, and we usually write $s \circ t$ instead of $\circ(s, t)$ (again we may have to write $(s \circ t)$ to prevent ambiguities),
- a nullary function symbol e.

The signature for groups extends the signature for monoids with

- a unary function symbol inv, and we usually write s^{-1} instead of $inv(s)$.

1.3 Contexts and Substitutions

Free and Bound Variables Bottom-up induction on syntax trees is the most important induction for logics and similar languages. It is often just called "induction on the grammar", or "induction on the language", or even just "induction". The following definitions are done in this way. The following definitions are for FOLEQ; the analogues for FOL are obtained by deleting the case for $\doteq$.

Definition 1.12 (Bound Variables)**.** We define the set $BV(w)$ of *bound variables* of a word w — i.e., a formula or a term — as follows.

- for terms:
  - $BV(x) = \varnothing$,
  - $BV(f(t_1, \ldots, t_n)) = BV(t_1) \cup \ldots \cup BV(t_n)$,
- for formulas:
  - $BV(p(t_1, \ldots, t_n)) = BV(t_1) \cup \ldots \cup BV(t_n)$,
  - $BV(t_1 \doteq t_2) = BV(t_1) \cup BV(t_2)$,
  - $BV(\mathit{true}) = \varnothing$,
  - $BV(\mathit{false}) = \varnothing$,
  - $BV(F \wedge G) = BV(F) \cup BV(G)$ and accordingly for the other connectives,
  - $BV(\forall x\ F) = BV(F) \cup \{x\}$ and accordingly for the other quantifier.

The opposite of bound variables are the free variables.

Definition 1.13 (Free Variables)**.** We define the set $FV(w)$ of *free variables* of a word w — i.e., a formula or a term — as follows.

- for terms:
  - $FV(x) = \{x\}$,
  - $FV(f(t_1, \ldots, t_n)) = FV(t_1) \cup \ldots \cup FV(t_n)$,
- for formulas:
  - $FV(p(t_1, \ldots, t_n)) = FV(t_1) \cup \ldots \cup FV(t_n)$,
  - $FV(t_1 \doteq t_2)) = FV(t_1) \cup FV(t_2)$,
  - $FV(\mathit{true}) = \varnothing$,
  - $FV(\mathit{false}) = \varnothing$,
  - $FV(F \wedge G) = FV(F) \cup FV(G)$ and accordingly for the other connectives,
  - $FV(\forall x\ F) = FV(F) \setminus \{x\}$ and accordingly for the other quantifier.

Remark 1.14. Note that:

- $x \in BV(F)$ iff there is a $\forall x$ or a $\exists x$ in F. We say that, e.g., $\forall x\ F$ binds the variable x in F. $\forall$ and $\exists$ are called *binders*.
- For a term t, we always have $BV(t) = \varnothing$ because terms cannot contain binders. This is a special property of FOL and FOLEQ — for example, lambda calculus is a language where terms may contain binders (namely the λ binder).
- The sets $FV(F)$ and $BV(F)$ are always finite.
- The definitions of $BV(-)$ and $FV(-)$ only differ in two cases.
- If a variable x occurs in F, then $x \in BV(F)$ or $x \in FV(F)$ or both. For example, x occurs both free and bound in $p(x) \wedge \forall x\ q(x)$.

Remark 1.15. A Σ-formula/term with an empty set of free variables is called *closed* or a *ground formula/term*. Ground formulas are often called *sentences* and

If F is a formula with $FV(F) = \{x_1, \ldots, x_n\}$, then $\forall x_1\ \ldots\ \forall x_n\ F$ and $\exists x_1\ \ldots\ \exists x_n\ F$ are closed formulas. We call them the *universal closure* and the *existential closure* of F.

Example 1.16. Here are some examples for free and bound variables:

- For the formula $F = x \circ (y \circ z) \doteq (x \circ y) \circ z$ over the signature of monoids, we have $BV(F) = \varnothing$ and $FV(F) = \{x, y, z\}$. Its universal closure is $\forall x\, \forall y\, \forall z\, x \circ (y \circ z) \doteq (x \circ y) \circ z$.
- There are not so many ground terms over the signature of monoids: They are e, $e \circ e$, $(e \circ e) \circ e$, $e \circ (e \circ e)$, and so on.
- If a FOL- or FOLEQ-signature has no constant symbols, then there are no ground terms.
- If a FOL-signature has a unary predicate symbol p, then $\exists x\, p(x)$ is a (non-boring) ground formula.
- If a FOLEQ-signature has a unary function symbol f, then $\exists x\, x \doteq f(x)$ is a (non-boring) ground formula.

The last two examples show that even with only one function or predicate symbol, FOL and FOLEQ already offer a lot of interesting formulas to study.

Contexts Above we defined the syntax FOLEQ by using a context-free grammar. This grammar produced all formulas, i.e., formulas with any number of free variables. This is often not desirable because we usually only care about closed formulas.

However, the language of closed formulas is not context-free. This is easy to see: Every binder $\forall x\, F$ declares a variable x; this variable is *local* in the sense that it has a certain *scope* and should only occur within its scope; for example, in $G \wedge \forall x\, F$, the scope of x is F (but not G). In general, all languages with such declarations are not context-free. Therefore, we can only give a context-free grammar for formulas but not for closed formulas.

This is the main reason why we use formulas with free variables at all: We can write down a context-free grammar for them, and such a grammar gives us the powerful and simple induction principle of induction on syntax trees.

In general, we use contexts to keep track of the free variables:

Definition 1.17 (Context)**.** Let Σ be a signature. A Σ-*context* Γ is a finite set of variables.

We define

$$\Gamma \vdash_\Sigma F : \texttt{form} \quad \text{iff} \quad F \in Form^{FOLEQ}(\Sigma) \text{ and } FV(F) \subseteq \Gamma$$

If $\Gamma \vdash_\Sigma F : \texttt{form}$, we say that F is a Σ-formula in context Γ. If $\Gamma = \varnothing$, we write $\vdash_\Sigma F : \texttt{form}$.

Finally, we define

$$\mathbf{Sen}^{FOLEQ}(\Sigma) = \{F \mid \vdash_\Sigma F : \texttt{form}\}$$

Example 1.18. Let Σ be the signature of monoids. The formula $x \circ (y \circ z) \doteq (x \circ y) \circ z$ is a formula in the context $\{x, y, z\}$:

$$x, y, z \vdash_\Sigma x \circ (y \circ z) \doteq (x \circ y) \circ z : \texttt{form}$$

It is also a formula in any bigger context:

$$x, x', y, z, y' \vdash_\Sigma x \circ (y \circ z) \doteq (x \circ y) \circ z : \texttt{form}$$

Substitutions We can think of free variables as holes or placeholder that can be filled with arbitrary terms. This filling operation is the task of substitutions.

Definition 1.19 (Substitution)**.** Let Σ be a signature. A *substitution* from a Σ-context Γ to a Σ-context Γ' is a mapping from Γ to Σ-terms in context Γ'.

Notation 1.20. If γ is a substitution from Γ to Γ', we write $\gamma : \Gamma \to \Gamma'$. If $\Gamma = \{x_1, \ldots, x_n\}$ and $\gamma(x_i) = t_i$, we write γ as $[x_1/t_1, \ldots, x_n/t_n]$.

Remark: It is not so common to use contexts when talking about FOL and FOLEQ. When you study additional literature, you will probably not find them. The reason is that — contrary to other logics — FOL and FOLEQ

are so simple that contexts are not needed. However, contexts make a lot of things much clearer and more elegant, and a good computer scientist should understand them. Also, some literature writes substitutions the other way around: t/x instead of x/t.

Example 1.21. Let Σ be the signature of monoids.

- If $\Gamma = \{x, y, z\}$ and $\Gamma' = \{x, u, v, y\}$, then $[x/y, y/x, z/e \circ v]$ is a substitution from Γ to Γ'.
- The identity $[x_1/x_1, \ldots, x_n/x_n]$ is a substitution from the context $\{x_1, \ldots, x_n\}$ to itself.
- More generally, $[x_1/x_1, \ldots, x_n/x_n]$ is a substitution from the context $\{x_1, \ldots, x_n\}$ to any context Γ' with $\{x_1, \ldots, x_n\} \subseteq \Gamma'$.

The intuition behind a substitution $[x_1/t_1, \ldots, x_n/t_n]$ from Γ to Γ' is that every (free) variable x_i occurring in a formula/term over Γ is substituted with t_i thus creating a formula/term over Γ'. This is formalized in the following definition.

Definition 1.22 (Substitution application)**.** Let Σ be a signature and $\gamma : \Gamma \to \Gamma'$ a substitution between Σ-contexts. Then we define the *application* $\overline{\gamma}(-)$ of γ to a word as follows:

- terms:
  - $\overline{\gamma}(x) = \gamma(x)$,
  - $\overline{\gamma}(f(t_1, \ldots, t_n)) = f\big(\overline{\gamma}(t_1), \ldots, \overline{\gamma}(t_n)\big)$,
- formulas:
  - $\overline{\gamma}(p(t_1, \ldots, t_n)) = p\big(\overline{\gamma}(t_1), \ldots, \overline{\gamma}(t_n)\big)$,
  - $\overline{\gamma}(t_1 \doteq t_2) = \overline{\gamma}(t_1) \doteq \overline{\gamma}(t_2)$,
  - $\overline{\gamma}(\mathit{true}) = \mathit{true}$,
  - $\overline{\gamma}(\mathit{false}) = \mathit{false}$,
  - $\overline{\gamma}(F \wedge G) = \overline{\gamma}(F) \wedge \overline{\gamma}(G)$ and similarly for the other connectives,
  - $\overline{\gamma}(\forall x\ F) = \forall x\ \overline{\gamma^x}(F)$ and similarly for the other quantifier.

  Here γ^x is a substitution from $\Gamma \cup \{x\}$ to $\Gamma' \cup \{x\}$ defined by $\gamma^x(x) = x$ and $\gamma^x(y) = \gamma(y)$ for all variables y in Γ. (In the special case where $x \in \Gamma$, the former takes precedence.)

Notation 1.23. If F is a formula with $FV(F) = x_1, \ldots, x_n$, it is common to use the following abbreviation:

$$F(t_1, \ldots, t_n) \quad := \quad \overline{[x_1/t_1, \ldots, x_n/t_n]}(F).$$

And similarly, if t is a term with $FV(t) = x_1, \ldots, x_n$, then:

$$t(t_1, \ldots, t_n) \quad := \quad \overline{[x_1/t_1, \ldots, x_n/t_n]}(t).$$

Moreover, we use the following abbreviations

$$F[x_1/t_1, \ldots, x_n/t_n] \quad := \quad \overline{[x_1/t_1, \ldots, x_n/t_n]}(F).$$

$$t[x_1/t_1, \ldots, x_n/t_n] \quad := \quad \overline{[x_1/t_1, \ldots, x_n/t_n]}(t).$$

There is one tricky problem with substitutions. Assume $\gamma = [x/y] : \{x\} \to \{y\}$. Then $\overline{\gamma}(\forall y\ p(x, y)) = \forall y\ p(y, y)$. This is not intended: We want to obtain something like $\forall y'\ p(y, y')$, i.e., we want to distinguish the first y, which should stem from the context $\{y\}$ and the second one, which should stem from the binder $\forall y$. If a substitution makes two variables equal that are supposed to different, we speak of *variable capture*. In this example, we can avoid variable capture only by renaming y to y'. This is called *α-renaming*.

Definition 1.24 (α-renaming)**.** Replacing a (sub-)formula $\forall x\ F$ with $\forall x'\ F'$ where F' is like F but with all occurrences of x replaced with x' is called an *α-renaming* (from x to x'). The same holds for $\exists$ instead of $\forall$.

More generally, we can always avoid variable capture as follows: Before applying the substitution $\gamma : \Gamma \to \Gamma'$ to the formula F, we use α-renaming to replace all variables $x \in BV(F) \cap \Gamma'$ with other variables. If we want to indicate that we do that, we speak of *capture-avoiding substitutions*.

Example 1.25. Let Σ be the signature of monoids. Let $\Gamma = \{x, y, z\}$ and $\Gamma' = \{x, u, v, y\}$, and $\gamma = [x/y, y/x, z/e \circ v]$ as above. Then

- $\overline{\gamma}\big(x \circ (y \circ z) \doteq (x \circ y) \circ z\big) \quad = \quad y \circ (x \circ (e \circ v)) \doteq (y \circ x) \circ (e \circ v)$,
- $\overline{\gamma}\big(\forall z\ x \circ (y \circ z) \doteq (x \circ y) \circ z\big) \quad = \quad \forall z\ y \circ (x \circ z) \doteq (y \circ x) \circ z$,
- $\overline{\gamma}\big(\forall y\ x \circ (y \circ z) \doteq (x \circ y) \circ z\big) \quad = \quad \forall y\ y \circ (y \circ (e \circ v)) \doteq (y \circ y) \circ (e \circ v)$
  (variable capture)
- $\overline{\gamma}\big(\forall y'\ x \circ (y' \circ z) \doteq (x \circ y') \circ z\big) \quad = \quad \forall y'\ y \circ (y' \circ (e \circ v)) \doteq (y \circ y') \circ (e \circ v)$
  (capture-avoiding variant of the above after α-renaming of y to y')

Eventually we can state the main property of substitutions:

Theorem 1.26 (Closure under Substitution)**.** *Let Σ be a signature and $\gamma : \Gamma \to \Gamma'$ a substitution between Σ-contexts. Then:*

- *if* $\Gamma \vdash_\Sigma t : \texttt{term}$*, then* $\Gamma' \vdash_\Sigma \overline{\gamma}(t) : \texttt{term}$*,*
- *if* $\Gamma \vdash_\Sigma F : \texttt{form}$*, then* $\Gamma' \vdash_\Sigma \overline{\gamma}(F) : \texttt{form}$*,*

where, if necessary, the substitution application uses α-renaming in F such that variable capture is avoided.

Proof. This is left as an exercise. □

1.4 An Abstract Definition of the Syntax of a Logic

Above we have seen the signatures and formulas of three logics: PL, FOL, and FOLEQ. And there are many more logics: Researchers have developed all kinds of logics for different purposes. For example, two general ways how to obtain new logics from old ones are the following.

- Changing the signatures: for example, ALGEQ (algebraic equational logic) arises from FOLEQ by only using signatures without predicate symbols.
- Changing the formulas: for example, HORNEQ (equational Horn logic) arises from FOLEQ by only allowing formulas that look like this:
  $$\forall x_1 \ldots \forall x_m\ ((F_1 \wedge \ldots \wedge F_n) \to F)$$
  where all the $F_1, \ldots, F_n, F$ are atomic (i.e., they do not contain proper subformulas).

Therefore, it is important to study the abstract properties of logics so that we can work with all these logics at once. Otherwise, every definition or theorem would have to be repeated for every logic.

Therefore, we introduce the following definition.

Definition 1.27. A *logic syntax* is a pair $(\mathbf{Sig}, \mathbf{Sen})$ where $\mathbf{Sig}$ is a collection of objects — called the *signatures* — and $\mathbf{Sen}$ is a mapping from $\mathbf{Sig}$ to sets, i.e., if $\Sigma \in \mathbf{Sig}$, then $\mathbf{Sen}(\Sigma)$ is a set. The elements of $\mathbf{Sen}(\Sigma)$ are called the *Σ-sentences*.

Example 1.28. Now we can say:

- Let $\mathbf{Sig}^{PL}$ be the collection of PL-signatures, and $\mathbf{Sen}^{PL}$ be the mapping that maps a PL-signature to its set of PL-formulas. Then $(\mathbf{Sig}^{PL}, \mathbf{Sen}^{PL})$ is a logic syntax.
- Let $\mathbf{Sig}^{FOL}$ be the collection of FOL-signatures, and $\mathbf{Sen}^{FOL}$ be the mapping that maps a FOL-signature to its set of FOL-sentences. Then $(\mathbf{Sig}^{FOL}, \mathbf{Sen}^{FOL})$ is a logic syntax.
- The logic syntax $(\mathbf{Sig}^{FOLEQ}, \mathbf{Sen}^{FOLEQ})$ is defined similarly to $(\mathbf{Sig}^{FOL}, \mathbf{Sen}^{FOL})$.

When defining logics (or related formalisms) in this way, we typically end up with the following situation:

- There is a collection of signatures Σ.
- For every signature, there is a collection of contexts $Con(\Sigma)$.
- For every signature Σ and every context Γ, there is the set of formulas $\Gamma \vdash_\Sigma F : \texttt{form}$.
- For a fixed signature Σ, we can use substitutions $\gamma : \Gamma \to \Gamma'$ between two Σ-contexts Γ and Γ'.
- Such a substitution $\gamma : \Gamma \to \Gamma'$ induces a mapping from formulas $\Gamma \vdash_\Sigma F : \texttt{form}$ to formulas $\Gamma' \vdash_\Sigma \overline{\gamma}(F) : \texttt{form}$.

For example, this is the case for FOL and FOLEQ. FOL and FOLEQ have the special property that for all signatures Σ, the contexts are simply finite sets of variables. In other logics, the contexts can be more complex.

1.5 Theories

The above definition of logic syntax comes in very handy when we define theories: Theories are defined in exactly the same way for every logic. With our definition of logic syntax, we can handle all logics at once.

Definition 1.29 (Theories)**.** Let $L = (\mathbf{Sig}, \mathbf{Sen})$ be a logic syntax. Then an *L-theory* is pair (Σ, Θ) where $\Sigma \in \mathbf{Sig}$ and $\Theta \subseteq \mathbf{Sen}(\Sigma)$. The formulas in Θ are called the *axioms* of the theory.

The most important logic, for which theories are used, is FOLEQ. A FOLEQ-theory is a pair of a FOLEQ signature and a set of FOLEQ-formulas. Let us look at some examples of FOLEQ-theories.

Example 1.30 (Set Theory). Below we will use the abbreviation $F \leftrightarrow G$ for $(F \to G) \wedge (G \to F)$. The theory of set theory consists of the above-mentioned signature for set theory and the following axioms:

- axiom of equality (also called extensionality, intuitively: Two sets are equal if they have the same elements.): $\forall X\ \forall Y\ (X \doteq Y \leftrightarrow \forall z\ (z \in X \leftrightarrow z \in Y))$,
- axiom characterizing the empty set: $\forall x\ \neg x \in \varnothing$,
- axioms characterizing the unordered pair: $\forall x\ \forall y\ \{x, y\} \doteq \{y, x\}$ and $\forall x\ \forall y\ x \in \{x, y\}$,
- axiom characterizing the union: $\forall X\ \forall z\ (z \in \bigcup X \leftrightarrow \exists x\ (x \in X \wedge z \in x))$,
- axiom characterizing the power set: $\forall X\ \forall z\ (z \in Pow(X) \leftrightarrow z \subseteq X)$,
- some other axioms, which would go beyond the scope of this lecture. A nice overview is given at http://en.wikipedia.org/wiki/Zermelo%E2%80%93Fraenkel_set_theory. However, there the variant of set theory is used where $\in$ is the only function/predicate symbol.

As said above, this formulation of set theory goes back to works by Zermelo and Fraenkel in the 1920s [Zer08, Fra22]. However, it was not originally written in FOLEQ. In fact, FOLEQ was not even fully understood yet at the time. It has undergone various changes in modern mathematics, and today different versions of set theory are used.

Example 1.31 (Monoids). The theory of monoids consists of the above-mentioned signature for monoids and the following axioms:

- associativity: $\forall x\ \forall y\ \forall z\ x \circ (y \circ z) \doteq (x \circ y) \circ z$,
- left-neutrality: $\forall x\ e \circ x \doteq x$,
- right-neutrality: $\forall x\ x \circ e \doteq x$.

Example 1.32 (Groups). The theory of groups consists of the above-mentioned signature for groups, all the axioms of the theory of monoids, and the following axioms:

- left-inverseness: $\forall x\ x \circ x^{-1} \doteq e$,
- right-inverseness: $\forall x\ x^{-1} \circ x \doteq e$.

Example 1.33 (Natural numbers). The theory of natural numbers is a tricky one, which we will come back to later (see Ex. 3.44 and 6.45). For now we can define it like this:

- a nullary function symbol 0,
- a unary function symbol *succ*,
- an axiom for the injectivity of *succ*: $\forall x\ \forall y\ (\mathit{succ}(x) \doteq \mathit{succ}(y) \to x \doteq y)$,
- an axiom that makes 0 the starting point: $\neg\exists x\ 0 \doteq \mathit{succ}(x)$.
- a set of axioms for induction: For every formula F with $FV(F) = \{x\}$, the axiom

$$\big(F(0) \wedge \forall\, x\big(F(x) \to F(\mathit{succ}(x))\big)\big) \to \forall x\ F(x).$$

These are Peano's axioms from 1889 [Pea89]. As for set theory, they were written at a time when researchers only began to study the notions of logic and formal syntax. In particular, Peano did not know FOLEQ as we do today.

Example 1.34 (Natural numbers). A variant of the theory of natural numbers – usually called Peano arithmetic – extends Ex. 1.33 with:

- binary function symbols $+$ and $\cdot$,
- two axioms that define addition inductively,
- two axioms that define multiplication inductively.

Chapter 2

Context-Sensitive Well-Formedness

2.1 Inference Systems

Definition 2.1 (Inference System/Calculus)**.** An *inference system* consists of a set of *judgments* and a set of (inference) *rules*. A rule must be of the form

$$\frac{J_1 \quad \dots \quad J_n}{J} R$$

where $J, J_1, \dots, J_n$ are judgments. R is the (optional) name, J the *conclusion*, and $J_1, \dots, J_n$ the *hypotheses* of the rule. If $n = 0$, the rule is called an *axiom*.

The intuition of a rule as above is that we can derive/prove/conclude that J holds if we have already derived/proved/-concluded that $J_1, \dots, J_n$ hold.

Definition 2.2 (Derivation)**.** A *derivation* in an inference system is a tree in which

- every node N is labelled with a judgment $J(N)$,
- for every node N with children $N_1, \dots, N_r$, the inference system contains the rule

$$\frac{J(N_1) \quad \dots \quad J(N_r)}{J(N)}$$

A judgment J *holds* iff there is a derivation whose root is labelled with J.

Remark 2.3. Note that the leaves of a derivation must be labelled with axioms.

The intuition of a derivation is that it is the proof/evidence/justification of a judgment. Judgements that occur as the root of a derivation are said to be established/derived/proved, to hold, or to be true.

The intuition of an inference system is that it defines which judgments hold.

Remark 2.4 (Relation to Parse Trees). A derivation over an inference system I is the same as a parse tree over the grammar containing one production

$$D ::= R(\underbrace{D, \dots, D}_{n})$$

for every rule R of I with n hypotheses.

2.2 Inference Systems for Well-Formed Syntax

Logical languages are usually context-sensitive. Therefore, logic has developed a specific method for defining context-sensitive languages, which uses a pair of a grammar and an inference system. The inference system uses a

context.

As a first inference system, we give a supplementary definition to Def. 1.5.

Definition 2.5. For the syntax of FOL, we use the following two judgments:

- If $\Gamma = \{x_1, \ldots, x_n\}$ is a Σ-context and t is a term, the judgment

$$\Gamma \vdash_\Sigma t : \texttt{term} \qquad \text{or} \qquad x_1, \ldots, x_n \vdash_\Sigma t : \texttt{term}$$

  (intended to mean that t is a well-formed Σ-term in context Γ).

- If $\Gamma = \{x_1, \ldots, x_n\}$ is a Σ-context and F is a formula, the judgment

$$\Gamma \vdash_\Sigma F : \texttt{form} \qquad \text{or} \qquad x_1, \ldots, x_n \vdash_\Sigma F : \texttt{form}$$

  (intended to mean that F is a well-formed Σ-formula in context Γ).

The inference rules are given in Fig. 2.1.

$$\frac{\Gamma \vdash_\Sigma t_1 : \texttt{term} \quad \ldots \quad \Gamma \vdash_\Sigma t_n : \texttt{term} \quad f \in \Sigma_f \quad ar(f) = n}{\Gamma \vdash_\Sigma f(t_1, \ldots, f_n) : \texttt{term}} \qquad \frac{x \in \Gamma}{\Gamma \vdash_\Sigma x : \texttt{term}}$$

$$\frac{\Gamma \vdash_\Sigma t_1 : \texttt{term} \quad \ldots \quad \Gamma \vdash_\Sigma t_n : \texttt{term} \quad p \in \Sigma_p \quad ar(p) = n}{\Gamma \vdash_\Sigma p(t_1, \ldots, f_n) : \texttt{form}} \qquad \frac{\Gamma \vdash_\Sigma t_1 : \texttt{term} \quad \Gamma \vdash_\Sigma t_2 : \texttt{term}}{\Gamma \vdash_\Sigma t_1 \doteq t_2 : \texttt{form}}$$

$$\frac{\Gamma \vdash_\Sigma F : \texttt{form} \quad \Gamma \vdash_\Sigma G : \texttt{form}}{\Gamma \vdash_\Sigma F \wedge G : \texttt{form}} \qquad \frac{\Gamma \vdash_\Sigma F : \texttt{form} \quad \Gamma \vdash_\Sigma G : \texttt{form}}{\Gamma \vdash_\Sigma F \vee G : \texttt{form}} \qquad \frac{\Gamma \vdash_\Sigma F : \texttt{form} \quad \Gamma \vdash_\Sigma G : \texttt{form}}{\Gamma \vdash_\Sigma F \to G : \texttt{form}}$$

$$\frac{\Gamma \vdash_\Sigma F : \texttt{form}}{\Gamma \vdash_\Sigma \neg F : \texttt{form}} \qquad \frac{\Gamma, x \vdash_\Sigma F : \texttt{form}}{\Gamma \vdash_\Sigma \forall x\, F : \texttt{form}} \qquad \frac{\Gamma, x \vdash_\Sigma F : \texttt{form}}{\Gamma \vdash_\Sigma \exists x\, F : \texttt{form}}$$

Figure 2.1: Syntax of FOL

Lemma 2.6. *Assume a FOL-signature Σ and a Σ-context Γ.*

- $\Gamma \vdash_\Sigma t : \texttt{term}$ *holds iff* *t is a well-formed term over Σ with $FV(t) \subseteq \Gamma$.*
- $\Gamma \vdash_\Sigma F : \texttt{form}$ *holds iff* *F is a well-formed formula over Σ with $FV(F) \subseteq \Gamma$.*

Proof. Exercise. □

Chapter 3

Model Theory

In the previous chapter, we saw the syntax of several logics. Even though the symbols used in the formulas had an intuitive meaning (We all knew that $\wedge$ is supposed to mean "and".), we avoided these intuitions. In this chapter, we will define formally the meaning of the symbols and the formulas. Instead of "meaning", it is also common to say *interpretation* or *semantics*.

The semantics of a context-free language is usually defined as a function from the syntax to some other language. This function is defined by induction on syntax trees. The "other language" is called the *meta-language*. For programming languages, the meta-language is often natural language. For example, the semantics of Java is given in various big books called something like "Java language specification". Another meta-language used to define the semantics of a programming language is assembler (or byte code in the case of Java). In that case the above-mentioned function is implemented in the compiler.

A crucial characteristic of logics is that the meta-language is a formal language as well, namely mathematics. There are many names for the function that maps syntax to semantics in the literature. We will use the function $[\![-]\!]$. Thus, $[\![w]\!]$ is the semantics of w.

Based on a context-free grammar for the syntax, $[\![-]\!]$ can be described as follows:

- Every non-terminal A of the syntax (e.g., `form` and `term`) is mapped to a set $[\![A]\!]$.
- Every word w (e.g., a formula or term, respectively) derived from A is mapped to an element $[\![w]\!] \in [\![A]\!]$.
- The latter is defined by induction on the syntax tree that derives w from A.

3.1 Semantics of Propositional Logic

3.1.1 The Standard Semantics

First we interpret the single non-terminal:

$$[\![\texttt{form}]\!] = \{0, 1\}$$

The intuition here is that for all formulas F, we have that $[\![F]\!] = 1$ denotes "F is true", and $[\![F]\!] = 0$ denotes "F is false". The elements 0 and 1 are called the *truth values*. Instead of $\{0, 1\}$, it is also common to use the sets $\{\bot, \top\}$ (bottom and top) or $\{F, T\}$.

Then we need a mapping $[\![-]\!] : \mathbf{Sen}^{PL}(\Sigma) \to \{0, 1\}$ for all signatures Σ. The definition is by induction on syntax trees, and we need one case for every production. The straightforward approach looks like this:

- $[\![\mathit{true}]\!] = 1$,
- $[\![\mathit{false}]\!] = 0$,
- $[\![F \wedge G]\!] = \min\{[\![F]\!], [\![G]\!]\} = \begin{cases} 1 & \text{if } [\![F]\!] = 1 \text{ and } [\![G]\!] = 1 \\ 0 & \text{otherwise} \end{cases}$,
- $[\![F \vee G]\!] = \max\{[\![F]\!], [\![G]\!]\} = \begin{cases} 1 & \text{if } [\![F]\!] = 1 \text{ or } [\![G]\!] = 1 \\ 0 & \text{otherwise} \end{cases}$,

- $[\![F \to G]\!] = \min\{[\![G]\!] - [\![F]\!], 0\} + 1 = \begin{cases} 1 & \text{if } [\![F]\!] \leq [\![G]\!] \\ 0 & \text{otherwise} \end{cases}$,
- $[\![\neg F]\!] = 1 - [\![F]\!] = \begin{cases} 1 & \text{if } [\![F]\!] = 0 \\ 0 & \text{otherwise} \end{cases}$,
- for $p \in \Sigma$: $[\![p]\!] = ???$

The above definition shows two things: Firstly, $\wedge$, $\vee$, $\to$, and $\neg$ can be interpreted naturally as "and", "or", "if ... then", and "not", respectively. Secondly, the boolean variables $p \in \Sigma$ cannot be interpreted naturally. This was to be expected: The boolean variables represent arbitrary statements, and of course arbitrary statements can be true or false depending on the situation.

Therefore, we cannot simply define a mapping $[\![-]\!] : \mathbf{Sen}(\Sigma) \to \{0, 1\}$. Instead, we introduce the concept of models.

Definition 3.1 (Models)**.** A *model* of the PL-signature Σ is a mapping $\Sigma \to \{0, 1\}$. We denote the set of PL-models for Σ by $\mathbf{Mod}^{PL}(\Sigma)$.

The idea of models is that they collect all the cases that are missing in the inductive definition of the semantics. Those are all cases for production that refer to the signature. In the case of PL, there is only one such production, namely `form` $::= p$. Therefore, PL-models consists of one function that interprets all $p \in \Sigma$. The semantics is then defined *relative to a model*, i.e., a formula has different meanings depending on the model in which it is interpreted. Instead of "model", it is common to say *interpretation function* or *structure*.

Now we can define the semantics of PL as follows.

Definition 3.2 (Semantics of PL)**.** Assume a PL-signature Σ and a model $I \in \mathbf{Mod}^{PL}(\Sigma)$. Then we define the interpretation $[\![-]\!]^I : \mathbf{Sen}(\Sigma) \to \{0, 1\}$ of Σ-sentences in I as follows:

- $[\![\mathit{true}]\!]^I = 1$,
- $[\![\mathit{false}]\!]^I = 0$,
- $[\![F \wedge G]\!]^I = \min\{[\![F]\!]^I, [\![G]\!]^I\}$,
- $[\![F \vee G]\!]^I = \max\{[\![F]\!]^I, [\![G]\!]^I\}$,
- $[\![F \to G]\!]^I = \min\{[\![G]\!]^I - [\![F]\!]^I, 0\} + 1$,
- $[\![\neg F]\!]^I = 1 - [\![F]\!]^I$,
- for $p \in \Sigma$: $[\![p]\!]^I = I(p)$.

Notation 3.3 (Satisfaction). We also write

$$I \models^{PL}_{\Sigma} F \qquad \text{iff} \qquad [\![F]\!]^I = 1$$

where PL and Σ are often omitted if they are clear from the context.

We say that I satisfies F or F holds in I.

Example 3.4. Assume the PL-signature $\Sigma = \{p, q, r\}$. A model $I \in \mathbf{Mod}^{PL}(\Sigma)$ is given by $I(p) = I(q) = 1$ and $I(r) = 0$. Then we have:

- $[\![p \wedge \mathit{true}]\!]^I = 1$,
- $[\![\neg(p \vee q) \vee r]\!]^I = 0$.

For some sentences, the semantics does not depend on the model. For example, we have $[\![\mathit{true} \wedge \neg \mathit{false}]\!]^I = 1$ and $[\![p \vee \neg p]\!]^I = 1$ in every model I. The study of such sentences is a primary concern of logic. Thus, they get a special name in the following definition.

Definition 3.5. Assume a PL-signature Σ and a Σ-sentence F. F is called

- a *theorem* or *tautology* if $[\![F]\!]^I = 1$ for all models I,
- *satisfiable* if $[\![F]\!]^I = 1$ for some (maybe all) models I,
- *unsatisfiable* or *contradiction* if $[\![F]\!]^I = 0$ for all models I.

Remark 3.6. One of the most important problems of computer science is the following: Given a PL-formula F, decide whether F is satisfiable or not. This problem is so important that it has its own name: *SAT*. It is a good exercise at this point to solve it, i.e., to give an algorithm that answers the above question for input F.

3.1.2 Other Semantics for the PL syntax

[1]

It is important to realize that a logic has a syntax and a semantics. This means that it is possible that two different logics have the same logic syntax (i.e., the same signatures and the same sentences) but different semantics. The following logics are not extremely important per se. But they serve as examples to illustrate that the semantics chosen above for the logic PL is not the only possible choice.

Example 3.7 (Fuzzy Propositional Logic). Fuzzy propositional logic (FPL) has the same syntax as PL. But $[\![\texttt{form}]\!] = [0;1]$, and a model for the signature Σ is a mapping $\Sigma \to [0;1]$. In other words, all real numbers between 0 and 1 are possible truth values. The intuition is that $[\![F]\!]^I = 0.4$ is that F is 40 % true. Thus, fuzzy logics permits us to talk about degrees of truth, insecure truths, and probabilities. Fuzzy logics have proved very useful in machine controls where interpretations often change gradually: For example, the rule "If another robot is *close*, decelerate *a little*." can could be used to avoid collisions of mobile robots.

The definition

$$[\![F \wedge G]\!]^I = \begin{cases} 1 & \text{if } [\![F]\!]^I = [\![G]\!]^I = 1 \\ 0 & \text{otherwise} \end{cases}$$

does not make sense anymore for fuzzy logics. But the definition

$$[\![F \wedge G]\!]^I = \min\{[\![F]\!]^I, [\![G]\!]^I\}$$

is still appropriate. Similarly, the other cases of Def. 3.2 can be reused.

Thus, if $[\![F]\!]^I = 0.4$ and $[\![G]\!]^I = 0.3$, then $[\![F \wedge G]\!]^I = 0.3$. There are also other fuzzy logics that use

$$[\![F \wedge G]\!]^I = [\![F]\!]^I \cdot [\![G]\!]^I.$$

Then we would have $[\![F \wedge G]\!]^I = 0.12$

Example 3.8 (Paraconsistent Propositional Logic). In paraconsistent logic, we use the same syntax as for PL, but put $[\![\texttt{form}]\!] = \{0, 1, ?, !\}$. The intuition is that $[\![F]\!]^I = ?$ means that the interpretation of F is unknown. The intuition of $[\![F]\!]^I = !$ means that F is both true and false. Paraconsistent logics have proved useful when talking about knowledge about the world: Often we do not know whether something is true or false, or we have arguments in favor of and against a certain conclusion.

3.2 Semantics of First-Order Logic

In principle, the semantics of FOLEQ and FOL is defined in the same way as the semantics of PL. The difference is that now we have two non-terminals that we must interpret (`form` and `term`) and that the signature has function

[1] This section can be skipped.

and predicate symbols that need a semantics from the models. Again, we start with the straightforward approach without using a model and see how far we get. The cases where we get stuck and need the models are written in gray.

- $[\![\texttt{form}]\!] = \{0, 1\}$,
- $[\![\texttt{term}]\!] = \texttt{term}^I$ where $\texttt{term}^I$ is some set that must come from the model
- $[\![\mathit{true}]\!] = 1$,
- $[\![\mathit{false}]\!] = 0$,
- $[\![x]\!]$ = some element of $\texttt{term}^I$ that must come from the model or from somewhere else
- $[\![f(t_1, \ldots, t_n)]\!] = f^I([\![t_1]\!], \ldots, [\![t_n]\!])$ where f^I is some mapping that must come from the model
- $[\![p(t_1, \ldots, t_n)]\!] = p^I([\![t_1]\!], \ldots, [\![t_n]\!])$ where p^I is some mapping that must come from the model
- $[\![F \wedge G]\!] = \min\{[\![F]\!], [\![G]\!]\}$,
- $[\![F \vee G]\!] = \max\{[\![F]\!], [\![G]\!]\}$,
- $[\![F \to G]\!] = \min\{[\![G]\!] - [\![F]\!], 0\} + 1$,
- $[\![\neg F]\!] = 1 - [\![F]\!]$,
- $[\![s \doteq t]\!] = \begin{cases} 1 & \text{if } [\![s]\!] = [\![t]\!] \\ 0 & \text{otherwise} \end{cases}$
- $[\![\forall x\ F]\!] = \min\{[\![F]\!](u) \mid u \in \texttt{term}^I\} = \begin{cases} 1 & \text{if } [\![F]\!](u) = 1 \text{ for all } u \in \texttt{term}^I \\ 0 & \text{otherwise} \end{cases}$
  where $[\![F]\!](u)$ somehow means to evaluate $[\![F]\!]$ for the input u,
- $[\![\exists x\ F]\!] = \max\{[\![F]\!](u) \mid u \in \texttt{term}^I\} = \begin{cases} 1 & \text{if } [\![F]\!](u) = 1 \text{ for some } u \in \texttt{term}^I \\ 0 & \text{otherwise} \end{cases}$
  where $[\![F]\!](u)$ somehow means to evaluate $[\![F]\!]$ for the input u.

Inspecting the above definition attempt, we find that a model I must give us a set $\texttt{term}^I$, and functions f^I and p^I for all function symbols f and predicate symbols p. We also need a way to interpret variables. The latter will be taken care of by assignments. Thus, we define as follows.

Definition 3.9 (Models)**.** A *model* I of the FOLEQ-signature $\Sigma = (\Sigma_p, \Sigma_f, ar)$ is a mapping with domain $\{\texttt{term}\} \cup \Sigma_f \cup \Sigma_p$ such that

- $\texttt{term}^I$ is a non-empty set (called the *universe* or the *carrier set*),
- for every $f \in \Sigma_f$ with $ar(f) = n$: f^I is a mapping $f^I : (\texttt{term}^I)^n \to \texttt{term}^I$,
- for every $p \in \Sigma_p$ with $ar(p) = n$: p^I is a mapping $p^I : (\texttt{term}^I)^n \to \{0, 1\}$.

Here and in the sequel, we write $\texttt{term}^I$, f^I, p^I instead of $I(\texttt{term})$, $I(f)$, $I(p)$.

We denote the collection of Σ-models by $\mathbf{Mod}^{FOLEQ}(\Sigma)$.

The FOL-models are the same as the FOLEQ-models. This makes sense because FOL and FOLEQ also have the same signatures.

Definition 3.10 (Assignments)**.** Let Σ be a signature.

- An *assignment* α for a set of variables Γ and the Σ-model I is a mapping $\alpha : \Gamma \to \texttt{term}^I$.
- If $\Gamma = \{x_1, \ldots, x_n\}$ and $\alpha(x_i) = u_i$, we write α as $[x_1/u_1, \ldots, x_n/u_n]$.
- If α is an assignment for Γ, we write $[\alpha, x/u]$ for the assignment for $\Gamma \cup \{x\}$ that maps x to u and all other variables $y \in \Gamma$ to $\alpha(y)$.

Remark 3.11 (Substitutions and Assignments). Substitutions and assignments behave very similarly and have very similar notations. Assignments replace variables with semantic values of some model (i.e., elements of $\mathtt{term}^I$) in the same way in which substitutions replace variables with syntactic terms of some other context.

If we have a model and an assignment, we can define the semantics of FOLEQ-formulas and terms. The definition follows the inductive definition of the syntax.

Definition 3.12 (Semantics of FOLEQ)**.** Given

- a FOLEQ-signature Σ and
- a Σ-context Γ,

we define the interpretation $[\![w]\!]^{I,\alpha}$ of a formula or term w over Σ in context Γ

- in a Σ-model I and
- under an assignment α for Γ into I

as follows by induction on Γ and w.

- $[\![\mathtt{form}]\!]^{I,\alpha} = \{0,1\}$,
- $[\![\mathtt{term}]\!]^{I,\alpha} = \mathtt{term}^I$,
- $[\![\mathit{true}]\!]^{I,\alpha} = 1$,
- $[\![\mathit{false}]\!]^{I,\alpha} = 0$,
- $[\![x]\!]^{I,\alpha} = \alpha(x)$,
- $[\![f(t_1,\ldots,t_n)]\!]^{I,\alpha} = f^I([\![t_1]\!]^{I,\alpha},\ldots,[\![t_n]\!]^{I,\alpha})$
- $[\![p(t_1,\ldots,t_n)]\!]^{I,\alpha} = p^I([\![t_1]\!]^{I,\alpha},\ldots,[\![t_n]\!]^{I,\alpha})$
- $[\![F \wedge G]\!]^{I,\alpha} = \min\{[\![F]\!]^{I,\alpha}, [\![G]\!]^{I,\alpha}\}$,
- $[\![F \vee G]\!]^{I,\alpha} = \max\{[\![F]\!]^{I,\alpha}, [\![G]\!]^{I,\alpha}\}$,
- $[\![F \to G]\!]^{I,\alpha} = \min\{[\![G]\!]^{I,\alpha} - [\![F]\!]^{I,\alpha}, 0\} + 1$,
- $[\![\neg F]\!]^{I,\alpha} = 1 - [\![F]\!]^{I,\alpha}$,
- $[\![s \doteq t]\!]^{I,\alpha} = \begin{cases} 1 & \text{if } [\![s]\!]^{I,\alpha} = [\![t]\!]^{I,\alpha} \\ 0 & \text{otherwise} \end{cases}$,
- $[\![\forall x\ F]\!]^{I,\alpha} = \min\{[\![F]\!]^{I,[x/u,\alpha]} \mid u \in \mathtt{term}^I\}$,
- $[\![\exists x\ F]\!]^{I,\alpha} = \max\{[\![F]\!]^{I,[x/u,\alpha]} \mid u \in \mathtt{term}^I\}$.

In particular, for $\Gamma = \alpha = \varnothing$ and $F \in \mathbf{Sen}(\Sigma)$, we write $[\![F]\!]^I$ for the interpretation of F in I. Then we also write

$$I \models_\Sigma F \qquad \text{iff} \qquad [\![F]\!]^I = 1.$$

The semantics of FOL is the same as that of FOLEQ except that the case for $\doteq$ is dropped.

Example 3.13. Let Σ be the signature of monoids from above. A model $I \in \mathbf{Mod}^{FOLEQ}(\Sigma)$ consists of

- a set $\mathtt{term}^I$,
- an element $e^I \in \mathtt{term}^I$,
- a binary mapping $\circ^I : \mathtt{term}^I \times \mathtt{term}^I \to \mathtt{term}^I$.

Models of such small signatures are typically written as $(\mathtt{term}^I, e^I, \circ^I)$.

For example, we can say $I_1 = (\mathbb{Z}, 0, +)$ is a Σ-model. I_1 has the further properties that it satisfies all the axioms of the theory of monoids:

- associativity: $I_1 \models_\Sigma \forall x\ \forall y\ \forall z\ x \circ (y \circ z) \doteq (x \circ y) \circ z$,
- left-neutrality: $I_1 \models_\Sigma \forall x\ e \circ x \doteq x$,
- right-neutrality: $I_1 \models_\Sigma \forall x\ x \circ e \doteq x$.

There are lots of other Σ-models, for example, $I_2 = (\mathbb{N}, 0, -)$. While I_2 is a Σ-model, it does not satisfy all the axioms of the theory of monoids:

- associativity does not hold: $I_2 \not\models_\Sigma \forall x\ \forall y\ \forall z\ x \circ (y \circ z) \doteq (x \circ y) \circ z$,
- left-neutrality does not hold: $I_2 \not\models_\Sigma \forall x\ e \circ x \doteq x$,
- right-neutrality, however, holds: $I_2 \models_\Sigma \forall x\ x \circ e \doteq x$.

As an exercise, you should apply the definition of the semantics step by step to verify that the above statements about I_1 and I_2 are correct.

Summary We can summarize the syntax and semantics of *symbols* as follows:

Symbol	Syntax	Semantics
logical symbol (here: $\wedge$, $\vee$, $\to$, $\neg$, $\forall$, $\exists$, $\doteq$)	always present	fixed interpretation
function or predicate symbol	declared (globally) in signature	interpreted by model
variable	declared (locally) in context	interpreted by assignment

Similarly, for a Σ-model I and an assignment α for a Σ-context Γ, we can summarize the syntax and semantics of *composed expressions* as follows:

Expression	Syntax	Semantics
term t	$\Gamma \vdash_\Sigma t : \mathtt{term}$	$[\![t]\!]^{I,\alpha} \in [\![\mathtt{term}]\!]^I$
formula F	$\Gamma \vdash_\Sigma F : \mathtt{form}$	$[\![F]\!]^{I,\alpha} \in [\![\mathtt{form}]\!]^I = \{0, 1\}$

This can be interpreted as saying that $[\![-]\!]$ (recursively) translates every judgment holding about the syntax to a judgment holding about the semantics: Whenever $\Gamma \vdash_\Sigma t : \mathtt{term}$, then $[\![t]\!]^{I,\alpha} \in [\![\mathtt{term}]\!]^I$, and accordingly for formulas.

3.3 Semantics of Contexts and Substitution

[2]

A substitution γ lets us move terms and formulas from a context Γ to a context Γ'. Often we are in a situation where we know the interpretation of F in context Γ and want to compute the interpretation of $\overline{\gamma}(F)$ in context Γ'. Of course, this is straightforward: apply the substitution γ to F, then use the definition of the semantics to interpret $\overline{\gamma}(F)$. But this requires two inductions on the syntax, and that can be quite complicated if F is a long formula.

Imagine we have a sentence $\forall x\ F(x)$, and for a certain model I we have already computed $[\![F(x)]\!]^{I,\alpha}$ for all assignments α in order to determine whether $I \models_\Sigma \forall x\ F(x)$. This can be very hard because if $\mathtt{term}^I$ is infinite, there are infinitely many assignments. Now we want to determine whether $\forall y\ F(t(y))$. For that, we need to compute $[\![F(t(y))]\!]^{I,\alpha}$ for all assignments α. $F(t(y))$ arises from $F(x)$ by substituting x with $t(y)$. We want to use this fact and reuse the work we have already done.

This is possible by giving a semantics to substitutions and then studying how the interpretations of F, γ, and $\overline{\gamma}(F)$ are interrelated.

[2] This section can be skipped.

Definition 3.14 (Semantics of Contexts)**.** Assume a FOLEQ-signature Σ, a Σ-model I, and a Σ-context Γ. We define the interpretation of Γ in I as follows: $[\![\Gamma]\!]^I$ is the set of assignments for Γ into I.

Definition 3.15 (Semantics of Substitutions)**.** Assume a FOLEQ-signature Σ, a Σ-model I, and a Σ-substitution $\gamma : \Gamma \to \Gamma'$. We define the interpretation of γ in I as follows:

$$[\![\gamma]\!]^I : [\![\Gamma']\!]^I \to [\![\Gamma]\!]^I,$$

$$[\![\gamma]\!]^I : \alpha' \mapsto \alpha \text{ where } \alpha(x) = [\![\overline{\gamma}(x)]\!]^{I,\alpha'}.$$

This looks confusing. Let us "type-check" the definition, i.e., check whether $[\![\gamma]\!]^I$ is indeed a mapping from $[\![\Gamma']\!]^I$ to $[\![\Gamma]\!]^I$:

1. The input to $[\![\gamma]\!]^I$ is an assignment $\alpha' \in [\![\Gamma']\!]^I$. Thus, α' is an assignment for Γ' into I.
2. The output of $[\![\gamma]\!]^I$ is supposed to be an assignment $\alpha \in [\![\Gamma]\!]^I$. Thus, we have to prove that α is an assignment for Γ into I.
3. Thus, we have to prove that α is a mapping from Γ to $\mathtt{term}^I$. So let us assume $x \in \Gamma$. We have to prove that $\alpha(x) \in \mathtt{term}^I$.
   (a) $x \in \Gamma$ and $\gamma : \Gamma \to \Gamma'$. Therefore, $\overline{\gamma}(x)$ is a term in context Γ'.
   (b) We know α' is an assignment for Γ' into I.
   (c) Therefore, we can compute $[\![\overline{\gamma}(x)]\!]^{I,\alpha'}$. And the result is an element of $\mathtt{term}^I$.
   (d) Thus, we have proved $\alpha(x) \in \mathtt{term}^I$.
4. Thus, we have proved that $\alpha = [\![\gamma]\!]^I(\alpha')$ is an element of $[\![\Gamma]\!]^I$.

So the semantics of substitutions is indeed well-defined. It maps assignments for Γ' to assignments for Γ. Syntactically, a substitution $\gamma : \Gamma \to \Gamma'$ translates from Γ to Γ': By applying γ, we move Γ-terms and formulas to Γ'. Now we have seen that semantically γ is also a translation, but in the opposite direction: It maps assignments for Γ' to assignments for Γ. This is a very general effect that we find for all logics. Researchers have only fully understood it within the last decades. And understanding it has led to tremendous insights into the relation between syntax and semantics.

Using the interpretation of contexts and substitutions, we can obtain a different perspective on the interpretation of terms and formulas. Fix a signature Σ and a model I. Take formula F in context Γ. We cannot write $[\![F]\!]^I$ because F has free variables — we need an assignment for Γ, and then we have $[\![F]\!]^{I,\alpha} \in \{0,1\}$. In other words: If F is fixed, then every assignment α gives us an element of $\{0,1\}$. Thus, we can define

$$[\![F]\!]^I : [\![\Gamma]\!]^I \to \{0,1\},$$

$$[\![F]\!]^I : \alpha \mapsto [\![F]\!]^{I,\alpha}.$$

Similarly, we can define for a term t in context Γ:

$$[\![t]\!]^I : [\![\Gamma]\!]^I \to \mathtt{term}^I,$$

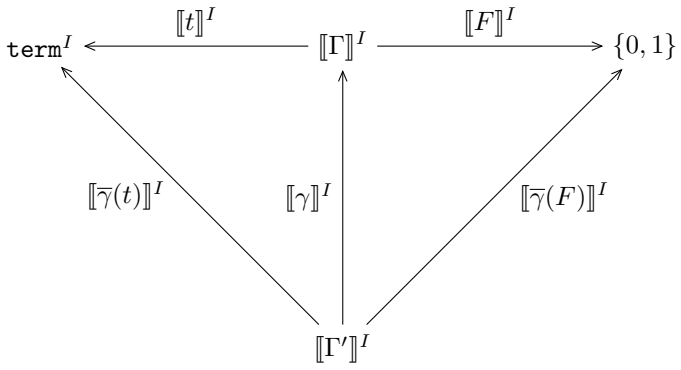
$$[\![t]\!]^I : \alpha \mapsto [\![t]\!]^{I,\alpha}.$$

Thus, we can think of the interpretation of a formula in context Γ as a mapping from the set of assignments (i.e., the interpretation of Γ) to $\{0,1\}$. And we can think of the interpretation of a term in context Γ as a mapping from the set of assignments to $\mathtt{term}^I$.

We can summarize this as follows:

Syntactically	Semantically
$\Gamma \vdash_\Sigma F : \mathtt{form}$	$[\![F]\!]^I : [\![\Gamma]\!]^I \to [\![\mathtt{form}]\!]^I = \{0,1\}$
$\Gamma \vdash_\Sigma t : \mathtt{term}$	$[\![t]\!]^I : [\![\Gamma]\!]^I \to [\![\mathtt{term}]\!]^I = \mathtt{term}^I$
$\gamma : \Gamma \to \Gamma'$	$[\![\gamma]\!]^I : [\![\Gamma']\!]^I \to [\![\Gamma]\!]^I$

If we look at this table for a while, we get a tempting idea. What if we compose maps as follows:



Could it be that the compositions yield exactly $[\![\overline{\gamma}(t)]\!]^I$ and $[\![\overline{\gamma}(F)]\!]^I$? Indeed they do.

Theorem 3.16 (Semantics of Substitution)**.** *Assume a FOLEQ-signature Σ and a Σ-model I. Then for every substitution $\gamma : \Gamma \to \Gamma'$ and every formula F in context Γ and every term t in context Γ:*

$$[\![\overline{\gamma}(t)]\!]^I = [\![t]\!]^I \circ [\![\gamma]\!]^I,$$

$$[\![\overline{\gamma}(F)]\!]^I = [\![F]\!]^I \circ [\![\gamma]\!]^I.$$

Proof. By context-sensitive induction on the syntax tree of t and F. □

Thus, the semantics of substitution is simply composition.

3.4 An Abstract Definition of the Model Theory of a Logic

Above we have seen the semantics for PL, FOL, and FOLEQ. In Sect. 1.4, we found that the syntax of a logic can be abstractly characterized by pairs $(\mathbf{Sig}, \mathbf{Sen})$, which we called a logic syntax. We can make a similar observation about the semantics.

Definition 3.17 (Model Theory)**.** A model theory for the logic syntax $(\mathbf{Sig}, \mathbf{Sen})$ is a pair $(\mathbf{Mod}, \models)$ such that

- for every $\Sigma \in \mathbf{Sig}$, we have that $\mathbf{Mod}(\Sigma)$ is a collection of objects, called the Σ-*models*,
- for every $\Sigma \in \mathbf{Sig}$, we have that $\models \subseteq \mathbf{Sen}(\Sigma) \times \mathbf{Mod}(\Sigma)$ is a binary relation, called the *satisfaction relation*.

If $I \models_\Sigma F$, we say "F holds in I" or "I satisfies F" or "I makes F true". We also write $I \not\models_\Sigma F$ for the opposite.

A four-tuple $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models)$ of a logic syntax $(\mathbf{Sig}, \mathbf{Sen})$ and a model theory $(\mathbf{Mod}, \models)$ for it is called a *model-theoretical* logic.

Remark 3.18. Model-theoretical logics were introduced as *institutions* in [GB92].

Example 3.19. Then we immediately get some example model theoretical logics:

- Propositional logic $PL = (\mathbf{Sig}^{PL}, \mathbf{Sen}^{PL}, \mathbf{Mod}^{PL}, \models^{PL})$.
- First-order logic with equality $FOLEQ = (\mathbf{Sig}^{FOLEQ}, \mathbf{Sen}^{FOLEQ}, \mathbf{Mod}^{FOLEQ}, \models^{FOLEQ})$. Note that $\mathbf{Sen}^{FOLEQ}(\Sigma)$ only contains the closed Σ-formulas; therefore, $[\![F]\!]^I$ only depends on the model I, and no assignments are needed when talking about $\models$.
- The logics FOL, ALGEQ, and HORNEQ are obtained from FOLEQ in the obvious way.

3.5 Theorems and Consequence (Model-Theoretically)

One of the most important advantages of the abstract definitions given by institutions is that the main definitions regarding theories can be done for all institutions at once.

Definition 3.20 (Models)**.** Given syntax and model theory $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models)$ and a theory $(\Sigma; \Theta)$. We define: A Σ-model I is a $(\Sigma; \Theta)$-model iff

$$I \models_\Sigma F \qquad \text{for all } F \in \Theta.$$

We write $\mathbf{Mod}(\Sigma, \Theta)$ for the collection of models of (Σ, Θ). In other words:

$$\mathbf{Mod}(\Sigma; \Theta) = \{I \in \mathbf{Mod}(\Sigma) \mid I \models_\Sigma F \text{ for all } F \in \Theta\}.$$

Then we are ready to make some far-reaching definitions:

Definition 3.21 (Monoids etc.)**.** We define:

- A model of the FOLEQ-theory from Ex. 1.31 is called a *monoid*.
- A model of the FOLEQ-theory from Ex. 1.32 is called a *group*.
- Commutative groups, rings, rings with 1, commutative rings, fields, etc. are all defined similarly using the respective signatures and axioms.

Example 3.22 (Fields). The theory of fields is given by the following signature

- binary function symbols $+$ and $\cdot$,
- unary function symbols $-$ and inv (where we write $inv(x)$ as x^{-1}),
- nullary functions symbols 0 and 1,

and the following axioms

- associativity of addition: $\forall x\ \forall y\ \forall z\ x + (y + z) \doteq (x + y) + z$,
- left-neutrality of zero: $\forall x\ 0 + x \doteq x$,
- right-neutrality of zero: $\forall x\ x + 0 \doteq x$,
- left-inverseness of subtraction: $\forall x\ (-x) + x \doteq 0$,
- right-inverseness of subtraction: $\forall x\ x + (-x) \doteq 0$,
- commutativity of addition: $\forall x\ \forall y\ x + y \doteq y + x$,
- associativity of multiplication: $\forall x\ \forall y\ \forall z\ x \cdot (y \cdot z) \doteq (x \cdot y) \cdot z$,
- left-neutrality of one: $\forall x\ 1 \cdot x \doteq x$,
- right-neutrality of one: $\forall x\ x \cdot 1 \doteq x$,
- left-inverseness of division: $\forall x\ (\neg x \doteq 0 \to x^{-1} \cdot x \doteq 1)$,
- right-inverseness of division: $\forall x\ (\neg x \doteq 0 \to x \cdot x^{-1} \doteq 1)$,
- commutativity of multiplication: $\forall x\ \forall y\ x \cdot y \doteq y \cdot x$,
- left-distributivity of multiplication over addition: $\forall x\ \forall y\ \forall z\ x \cdot (y + z) \doteq (x \cdot y) + (x \cdot z)$,
- right-distributivity of multiplication over addition: $\forall x\ \forall y\ \forall z\ (y + z) \cdot x \doteq (y \cdot x) + (z \cdot x)$.

A *field* is a model of this theory. Important fields are $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ with the usual functions.
Note that:

- We can define rings, rings with 1 (sometimes called unital rings or just rings), and commutative rings by dropping symbols and axioms from the theory of fields.
- In mathematics, *inv* is actually only a partial function: It is undefined for 0. In FOLEQ, we cannot talk about undefinedness. Therefore, we need a total function, and then the axioms need to say $\forall x\ (\neg x \doteq 0 \to \ldots)$.
- Right-neutrality, right-inverseness, and right-distributivity are actually redundant because of commutativity.

The area of mathematics called *algebra* is devoted to the study of the model collections $\mathbf{Mod}(\Sigma;\Theta)$ where $(\Sigma;\Theta)$ are the theories from above, in particular the theories of groups, rings, and fields.

Definition 3.23 (Theorems)**.** Given a syntax and model theory $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models)$ and a theory $(\Sigma;\Theta)$. We define:

- A Σ-sentence is a (model-theoretical) *theorem* of $(\Sigma;\Theta)$ if it is true in all (Σ,Θ)-models. In other words,
$$Thm(\Sigma;\Theta) = \{F \in \mathbf{Sen}(\Sigma) \mid I \models_\Sigma F \text{ for all } I \in \mathbf{Mod}(\Sigma,\Theta)\}.$$
- If $F \in Thm(\Sigma;\Theta)$, we also write
$$\Theta \models_\Sigma F.$$
Then we also say that F is a (model-theoretical) *consequence* of Θ.

Notation 3.24. Note that this means that the symbol $\models$ has two different but related meanings:

- $I \models_\Sigma F$ means that F is true in the model I. This is called *satisfaction*, and we say that I satisfies F.
- $\Theta \models_\Sigma F$ means that F is true in all models of (Σ,Θ). This is called *entailment*, and we say that Θ entails F.
- The relation between the two is:
$$\Theta \models_\Sigma F \qquad \text{iff} \qquad I \models_\Sigma F \quad \text{for all } I \quad \text{for which } I \models_\Sigma T \text{ for all } T \in \Theta.$$
Thus, intuitively, $\Theta \models_\Sigma F$ means "F holds in all models in which all sentences in Θ hold." Or "If a model satisfies all sentences in Θ, then it also satisfies F." Or even more intuitively: "If all sentences in Θ hold, then F holds."

Definition 3.25. A theory $(\Sigma;\Theta)$ is called *closed* if $\Theta = Thm(\Sigma;\Theta)$. $(\Sigma; Thm(\Sigma;\Theta))$ is a closed theory that arises by adding all theorems to Θ. This theory is called the *closure* of $(\Sigma;\Theta)$.

Theorem 3.26. *We have the following basic properties of theories:*

1. *If* $\Theta' \subseteq \Theta$*, then* $\mathbf{Mod}(\Sigma;\Theta') \supseteq \mathbf{Mod}(\Sigma,\Theta)$.
2. *If* $\Theta' \subseteq \Theta$*, then* $Thm(\Sigma;\Theta') \subseteq Thm(\Sigma,\Theta)$.
3. $F \in Thm(\Sigma;\Theta)$ *for every* Σ*-tautology* F.
4. $\Theta \subseteq Thm(\Sigma;\Theta)$.
5. $\mathbf{Mod}(\Sigma; Thm(\Sigma;\Theta)) = \mathbf{Mod}(\Sigma;\Theta)$.
6. $Thm(\Sigma; Thm(\Sigma;\Theta)) = Thm(\Sigma;\Theta)$.

Proof. 1. Exercise.

2. Exercise.

3. Special case of the second result: Put $\Theta' = \varnothing$.

4. Assume $F \in \Theta$ (*). We have to show $F \in \mathit{Thm}(\Sigma; \Theta)$.

   (a) By definition of *Thm*, we have to show $I \models_\Sigma F$ for every model I such that $I \in \mathbf{Mod}(\Sigma; \Theta)$.

   (b) Assume a model $I \in \mathbf{Mod}(\Sigma; \Theta)$ (**). We have to show $I \models_\Sigma F$.

      i. By (**) and the definition of **Mod**, we have that $I \models_\Sigma G$ for every $G \in \Theta$.

      ii. By (*), we can put $G = F$ and obtain $I \models_\Sigma F$.

      iii. Done.

   (c) Done.

5. The $\subseteq$ direction follows by combining the first and the fourth result.
   To prove the $\supseteq$ direction, we verbalize the involved sets:

   - $\mathbf{Mod}(\Sigma; \Theta)$: the models satisfying all sentences in Θ
   - $\mathit{Thm}(\Sigma; \Theta)$: the sentences satisfied by all models satisfying all sentences in Θ
   - $\mathbf{Mod}(\Sigma; \mathit{Thm}(\Sigma; \Theta))$: the models satisfying all sentences satisfied by all models satisfying all sentences in Θ

   Then it is easy to see that $\mathbf{Mod}(\Sigma; \mathit{Thm}(\Sigma; \Theta)) \supseteq \mathbf{Mod}(\Sigma; \Theta)$

6. This follows immediately using the definition of *Thm* and the fifth result.

□

Now we introduce some important notions for theories.

Definition 3.27. For a fixed theory $(\Sigma; \Theta)$, a sentence $F \in \mathbf{Sen}(\Sigma)$ is called

- a *theorem*, a *tautology*, or *valid* if it holds in all $(\Sigma; \Theta)$-models,
- a *contradiction* or *unsatisfiable* if it holds in no $(\Sigma; \Theta)$-model.

The words "tautology" and "valid" are especially common for the special case $\Theta = \varnothing$.

Lemma 3.28. *If a logic has negation, then F is a theorem iff $\neg F$ is a contradiction and F is a contradiction iff $\neg F$ is a theorem.*

Proof. Clear because F holds in a model iff $\neg F$ does not hold in it, and vice versa. □

Example 3.29. Examples for tautologies are *true*, $\neg$*false*, $(F \wedge G) \to (G \wedge F)$, or $\neg\neg F \to F$. There is a number of important and fairly obvious tautologies. A good exercise is to find some more and check that they really are tautologies. An important less obvious tautology is this: *false* $\to F$ for any F.

The most important examples for contradictions are *false*, $F \wedge \neg F$ for any F, and $\neg F$ for any tautology F.

Thus, a theory splits the sentences into three groups: theorems (true in all models), contradictions (true in no model), and the rest (true in some but not all models). Thus, only the sentences in the third group differ between models – they distinguish the models. Depending on the size of the three groups, we distinguish two special cases of theories:

Definition 3.30. A theory $(\Sigma; \Theta)$ is called

- *inconsistent* if all sentences are theorems, $\Theta \models_\Sigma F$ for every $F \in \mathbf{Sen}(\Sigma)$,
- *consistent* if it is not inconsistent, i.e., there is a sentence that is not a theorem,
- *complete* if every sentence is either a theorem or a contradiction, i.e., if for every $F \in \mathbf{Sen}(\Sigma)$ either $\Theta \models_\Sigma F$ or $\Theta \models_\Sigma \neg F$.

Inconsistent theories make all sentences theorems including, e.g., *false*. Thus, truth becomes meaningless. For example, a theory is inconsistent if it contains *false* or any other contradiction as an axiom or if it contains both F and $\neg F$ as axioms. An inconsistent theory cannot have a model. Precisely, we have:

Lemma 3.31. *The following are equivalent for a theory* $(\Sigma;\Theta)$ *in any logic:*

1. *It has no model.*
2. *It is inconsistent.*
3. *(If negation is interpreted as for PL:)* F *and* $\neg F$ *are theorems for some* F*.*
4. *(If falsity is interpreted as for PL:) false is a theorem.*

Proof. By circular implications:
1 implies 2: If there is no model, then every sentence is a theorem.
2 implies 3: If every sentence is a theorem, then at least F and $\neg F$ for some F.
3 implies 4: No model can satisfy F and $\neg F$. So every model that does also satisfies *false*.
4 implies 1: No model satisfies *false*. □

Complete theories make "half the sentences" theorems: For every F either F or $\neg F$ is a theorem, and the other one is a contradiction. Since the third group is empty, models have no freedom how to interpret the sentences. If we add one more sentence to a complete theory, it becomes inconsistent. Precisely, we have:

Lemma 3.32. *The following are equivalent for a theory* $(\Sigma;\Theta)$*:*

1. *It is complete.*
2. *There are models, and all models satisfy exactly the same formulas.*
3. *There is a model satisfying exactly the theorems of* $(\Sigma;\Theta)$*.*

Proof. By circular implications:
1 implies 2: If there were no models, then the theory would be inconsistent, but completeness implies consistency. These models must satisfy at least all the theorems of $(\Sigma;\Theta)$. Due to completeness, if some model satisfied one further sentence, that sentence would have to be a contradiction, which is impossible.
2 implies 3: Since all models satisfy the same sentences, those are the theorems of $(\Sigma;\Theta)$. Since there are models, we can pick any one of them, and that will satisfy exactly the theorems of $(\Sigma;\Theta)$.
3 implies 1: For any model I, the set $\{F \mid I \models_\Sigma F\}$ of sentences it satisfies is a complete theory. So $(\Sigma;\Theta)$ must be complete. □

Notation 3.33. We have defined the notions "consequence", "theorem", "contradiction", "(in)consistent", and "complete" model-theoretically. All have proof-theoretical analogues, see Def. 4.7 and 4.9. If we need to distinguish them, we will prefix M, e.g., we will say that a theory is M-consistent. See also Not. 4.11.

3.6 Specification and Abstract Data Types

The process of *specification* or *axiomatization* is the following: Given a class $\mathcal{M} \subseteq \mathbf{Mod}(\Sigma)$ of models, find a theory $\Theta \subseteq \mathbf{Sen}(\Sigma)$ such that $\mathcal{M}$ contains exactly the models of Θ.

Then we call the triple $(\Sigma, \Theta, \mathcal{M})$ an abstract data type.

3.6.1 Theoretical Foundation

[3]

Definition 3.34 (Model Class)**.** Assume a syntax-model theory pair $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models)$. A model class is a pair $(\Sigma; \mathcal{M})$ where $\Sigma \in \mathbf{Sig}$ and $\mathcal{M} \subseteq \mathbf{Mod}(\Sigma)$.

Model classes are the analogue to theories. Just like for theories, we can define two important operations.

[3] This section can be skipped.

Definition 3.35. For every model class $(\Sigma, \mathcal{M})$, we define the theory of $(\Sigma; \mathcal{M})$ by

$$Thy(\Sigma; \mathcal{M}) = \{F \in \mathbf{Sen}(\Sigma) \mid I \models_\Sigma F \text{ for all } I \in \mathcal{M}\}$$

and the closure of M by

$$ModC(\Sigma; \mathcal{M}) = \{I \in \mathbf{Mod}(\Sigma) \mid I \models_\Sigma F \text{ for all } F \in Thy(\Sigma; \mathcal{M})\}$$

Model classes satisfying $ModC(\Sigma; \mathcal{M}) = \mathcal{M}$ are called closed.

Notation 3.36. Let us now fix a signature Σ and use the following abbreviations:

- Θ^* for $\mathbf{Mod}(\Sigma; \Theta)$,
- $\mathcal{M}^*$ for $Thy(\Sigma; \mathcal{M})$,

Then we have the following mappings:

$$\begin{aligned}
\mathcal{P}(\mathbf{Sen}(\Sigma)) &\to \mathcal{P}(\mathbf{Mod}(\Sigma)), \quad & \Theta &\mapsto \Theta^* \\
\mathcal{P}(\mathbf{Mod}(\Sigma)) &\to \mathcal{P}(\mathbf{Sen}(\Sigma)), \quad & \mathcal{M} &\mapsto \mathcal{M}^* \\
\mathcal{P}(\mathbf{Sen}(\Sigma)) &\to \mathcal{P}(\mathbf{Sen}(\Sigma)), \quad & \Theta &\mapsto \Theta^{**} \\
\mathcal{P}(\mathbf{Mod}(\Sigma)) &\to \mathcal{P}(\mathbf{Mod}(\Sigma)), \quad & \mathcal{M} &\mapsto \mathcal{M}^{**}
\end{aligned}$$

and we have

$$\begin{aligned}
\Theta^{**} &= Thm(\Sigma; \Theta) \\
\mathcal{M}^{**} &= ModC(\Sigma; \mathcal{M})
\end{aligned}$$

Intuitively, $-^*$ turns a set of sentences into a collection of models (namely those satisfying all the sentences). It also maps the other way round: Every class of models is mapped to a set of sentences (namely those satisfied by all the models). Thus, $-^*$ maps back and forth between theories and model classes.

We can compose these mappings. If we start with a theory and go to model classes and back, i.e., we apply $-^*$ twice to $(\Sigma; \Theta)$, we obtain a new theory. And we already know this theory: It is $(\Sigma; Thm(\Sigma; \Theta))$. If we compose the other way round, we start with a model class $\mathcal{M}$, go to its theory, then take that theory's models. This composition also maps a model class $(\Sigma; \mathcal{M})$ to a larger one: It contains all the models that satisfy the same sentences as those in $\mathcal{M}$.

This is an instance of a very general concept called a Galois connection. Every binary relation $\rho \subseteq A \times B$ induces a Galois connection between $\mathcal{P}(A)$ and $\mathcal{P}(B)$. In our case $A = \mathbf{Sen}(\Sigma)$, $B = \mathbf{Mod}(\Sigma)$, and $\rho = \models_\Sigma$.

We have the following properties:

- $\Theta \subseteq \Theta'$ implies $\Theta^* \supseteq \Theta'^*$, and $\mathcal{M} \subseteq \mathcal{M}'$ implies $\mathcal{M}^* \supseteq \mathcal{M}'^*$,
- $\Theta \subseteq \Theta^{**}$ and $\mathcal{M} \subseteq \mathcal{M}^{**}$,
- $\Theta \subseteq \Theta'$ implies $\Theta^{**} \subseteq \Theta'^{**}$, and $\mathcal{M} \subseteq \mathcal{M}'$ implies $\mathcal{M}^{**} \subseteq \mathcal{M}'^{**}$,
- $\Theta^{**} = (\Theta^{**})^{**}$ and $\mathcal{M}^* = (\mathcal{M}^{**})^{**}$.
- Θ^* and $\mathcal{M}^*$ are closed.
- The closed theories are in bijection to the closed model classes.

Thus, specification means to find a Θ such that $\Theta^* = \mathcal{M}$. Often we are interested in the special case $\mathcal{M} = \{I\}$ for some intended model I.

It cannot always be possible to axiomatize a class of models: Θ^* is always a closed model class. So let us assume that $\mathcal{M}$ is closed. If $\mathcal{M}$ is closed, we can always find an axiomatization Θ by simply putting $\Theta := \mathcal{M}^*$.

But that does not help much: Θ should be as simple as possible. Ideally, Θ should be finite. But if Θ is infinite, it should at least be decidable (i.e., we should have an algorithm that tells us which axioms are in Θ). Thus, more precisely, specification means to find a simple Θ such that $\Theta^* = \mathcal{M}$.

Therefore, when we do specification, two things can go wrong:

- $\mathcal{M}$ is not closed, i.e., there is no appropriate Θ.
- There is a Θ but no simple one.

3.6.2 Examples

A lot of very important model classes can be specified in FOLEQ. We have already seen some in Ex. 3.22. Here are some more examples.

This approach of applying logic to systematically define data types in mathematics, specifically algebra, in terms of logic and model theory goes back to work by Robinson [Rob50].

Example 3.37 (Data types based on a binary function symbol). Some of the most important and prevalent data types of mathematics have finite axiomatizations in FOLEQ that are built as combinations of some simple function symbols and axioms. See Fig. 3.1.

symbol or axiom		magma	semigroup	commutative semigroup	monoid	commutative monoid	group	Abelian group	semilattice
$\circ$, binary function, written $s \circ t$	composition	✓	✓	✓	✓	✓	✓	✓	✓
e, nullary function	neutral element				✓	✓	✓	✓	
inv, unary function, written t^{-1}	inverse element						✓	✓	
$\forall x \forall y \forall z\ (x \circ y) \circ z \doteq x \circ (y \circ z)$	associative		✓	✓	✓	✓	✓	✓	✓
$\forall x\ (x \circ e \doteq x \wedge e \circ x \doteq x)$	neutral element				✓	✓	✓	✓	
$\forall x\ (x \circ x^{-1} \doteq e \wedge x^{-1} \circ x \doteq e)$	inverse element						✓	✓	
$\forall x \forall y\ x \circ y \doteq y \circ x$	commutative			✓		✓		✓	✓
$\forall x\ x \circ x \doteq x$	idempotency								✓

Figure 3.1: Data types for a binary function symbols

Example 3.38 (Models based on a binary function symbol). . The data types from Ex. 3.37 are important because their models occur all over mathematics and computer science.

We have the following models, where we write U for any number set, i.e., $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$:

- Addition of numbers: $(U, +, 0)$ is a commutative monoid. Additionally, if $U \neq \mathbb{N}$, $(U, +, 0, -)$ is an Abelian group.
- Addition of other objects: It is a widely used convention to name operations $+$ and 0 and $-$ if they form commutative monoids or Abelian groups, e.g., addition of vectors, matrices, polynomials.
- Multiplication of numbers: $(U, \cdot, 1)$ where $U \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}\}$ is a commutative monoid.
- Multiplication of other objects: It is a widely used convention to name operations $\cdot$ and 1 if they form monoids. However, these do not have to be commutative, e.g., multiplication of matrices.
- Maximum/minimum of numbers: $(\mathbb{N}, max, 0)$ is a commutative monoid. If $U \neq \mathbb{C}$, (U, max) and (U, min) are semilattices.
- Function composition: $(S^S, \circ, id_S)$ is a monoid; it is commutative iff $|S| \in \{0, 1\}$. If we only consider bijective functions, then inversion yields an inverse element and thus a group.
- Sets: $(\mathcal{P}(S), \cup, \varnothing)$ and $(\mathcal{P}(S), \cap, S)$ are commutative monoids and semilattices.

- Concatenation of words over an alphabet: $(\Sigma^*, \cdot, \varepsilon)$ is a monoid where $\cdot$ is concatenation of words. It is commutative iff $|\Sigma| \in \{0, 1\}$.
- Operations on languages over an alphabet: $(\mathcal{P}(\Sigma^*), \cdot, \{\varepsilon\})$ is a monoid where $\cdot$ is concatenation of languages.

Example 3.39 (Data types based on two binary function symbols). Fig. 3.2 gives datatypes that combine two of the datatypes from Ex. 3.39.

Additionally, we obtain

- the theory *ring without 1* by removing the 1 symbol and its axioms from the theory *ring*,
- the theory *lattice* by removing the 0 and 1 symbols and their axioms from the theory *bounded lattice*

symbol and/or axiom		semi-ring	ring	field	bounded lattice	Boolean lattice
binary functions $\circ$, $*$		✓				
associativity for $\circ$		✓				
associativity for $*$		✓				
nullary function 0 and neutral element axiom for $\circ$, 0		✓				
nullary function 1 and neutral element axiom for $*$, 1		✓				
commutativity for $\circ$		✓				
commutativity for $*$		optional[4]		✓		
$\forall x \forall y \forall z\ x * (y \circ z) \doteq (x * y) \circ (x * z)$ $\forall x \forall y \forall z\ (y \circ z) * x \doteq (y * x) \circ (z * x)$	distributivity		✓			✓
unary function $-$ and inverse element axiom for $\circ$, 0, $-$			✓			
$\forall x\ x * 0 \doteq 0 \wedge 0 * x \doteq 0$	annihilation	✓	T		✓	
$\forall x\ x \circ 1 \doteq 1 \wedge 1 \circ x \doteq 1$	(no common name)				✓	
idempotency for $\circ$					✓	
idempotency for $*$					✓	
$\forall x \forall y\ x * (x \circ y) \doteq x$ $\forall x \forall y\ x \circ (x * y) \doteq x$	absorption				✓	
unary function *inv*, written t^{-1}	inverse/complement			✓		✓
$\forall x\ \neg x \doteq 0 \to (x * x^{-1} \doteq 1 \wedge x^{-1} * x \doteq 1)$	multiplicative inverse			✓		
$\forall x\ x \circ x^{-1} \doteq 1 \wedge x^{-1} \circ x \doteq 1$ $\forall x\ x * x^{-1} \doteq 0 \wedge x^{-1} * x \doteq 0$	complement					✓

Figure 3.2: Data types for two binary function symbols (T marks theorems)

Example 3.40 (Models based on two binary function symbols). Models of the data types from Ex. 3.39 are often obtained by combining models from Ex. 3.38.

We write the models as $(\texttt{term}^I, \circ^I, 0^I, -^I, *^I, 1^I, \mathit{inv}^I)$ where we omit the components that are not applicable:

- Addition and multiplication: $(\mathbb{N}, +, 0, \cdot, 1)$ is a commutative semi-ring. $(\mathbb{Z}, +, 0, -, \cdot, 1)$ is a commutative ring. $(U, +, 0, -, \cdot, 1, ^{-1})$ are fields for $U \in \{\mathbb{Q}, \mathbb{R}, \mathbb{C}\}$.

[4] If present, the theory is called *commutative (semi-)ring*.

- Matrices: $(R^{n\times n}, +, 0, -, \cdot, 1)$ is a ring if R is. (Note that we have to use square matrices so that multiplication is a total function.) It is usually not commutative even if R is.
- Polynomials: $(R[X], +, 0, -, \cdot, 1)$ is a ring if R is; it is commutative if R is.
- Sets: $(\mathcal{P}(S), \cup, \varnothing, \cap, S, x \mapsto S \setminus x)$ is a Boolean lattice. Note that the special case $|S| = 1$ yields the usual 2-element Booleans.
- Languages: $(\mathcal{P}(\Sigma^*), \cup, \varnothing, \cdot, \{\varepsilon\})$ is a semiring.

Example 3.41 (Data types based on a binary predicate symbol). Similarly important data types use a binary predicate symbol and various combinations of axioms. See Fig. 3.3.

Of course, we usually use a different symbol than $\leq$ when a relation is symmetric, e.g., $\equiv$.

symbol or axiom		relation	equivalence relation	preorder	order, poset	total order	bounded order	lattice	dense order
$\leq$, written $s \leq t$	comparison	✓							
$\forall x\ x \leq x$	reflexive		✓						
$\forall x \forall y \forall z\ ((x \leq y \wedge y \leq z) \to x \leq z)$	transitive		✓						
$\forall x \forall y\ (x \leq y \to y \leq x)$	symmetric		✓						
$\forall x \forall y\ ((x \leq y \wedge y \leq x) \to x \doteq y)$	antisymmetric				✓				
$\forall x \forall y\ (x \leq y \vee y \leq x)$	total, linear					✓			
$\exists x \forall y\ x \leq y$	smallest element						✓		
$\exists x \forall y\ y \leq x$	greatest element						✓		
$\forall x \forall y \exists l\ (\mathit{below}(l, x, y) \wedge \forall l'\ (\mathit{below}(l', x, y) \to l' \leq l))$ where $\mathit{below}(z, x, y)$ abbreviates $z \leq x \wedge z \leq y$	greatest lower bound, infimum					T		✓	
$\forall x \forall y \exists u\ (\mathit{above}(u, x, y) \wedge \forall u'\ (\mathit{above}(u', x, y) \to u \leq u'))$ where $\mathit{above}(z, x, y)$ abbreviates $x \leq z \wedge y \leq z$	least upper bound, supremum					T		✓	
$\forall x \forall z\ (x < z \to \exists y\ (x < y \wedge y < z))$ where $x < y$ abbreviates $x \leq y \wedge \neg x \doteq y$	dense								✓

Figure 3.3: Data types for a binary predicate symbols (T marks theorems)

Example 3.42 (Models based on a binary predicate). Models of the data types from Ex. 3.41 are also very common. We use U as in Ex. 3.38.

- Size of numbers: $(U, \leq)$ is a total order for $U \neq \mathbb{C}$. It is dense for $U \in \{\mathbb{Q}, \mathbb{R}\}$. It has a smallest element if $U = \mathbb{N}$.
- Divisibility: $(\mathbb{N}, |)$ is an order. 1 is the smallest element and 0 the greatest element. Greatest lower bound and least upper bound exist and are called greatest common divisor and least common multiple.
- For any order (A, r), the inverse $(A, (u, v) \mapsto r(v, u))$ is an order, too. This yields $(U, \geq)$ for numbers.
- For any order (A, r), the irreflexive variant (A, r') where $r'(u, v)$ holds if $r(u, v)$ and $u \neq v$ is still transitive. This yields $(U, <)$ and $(U, >)$ for numbers.

- For any preorder $(A, \prec)$, we obtain an equivalence relation $(A, \equiv)$ by putting $u \equiv v$ iff $u \prec v$ and $v \prec u$. In that case, we obtain an order $(A/\equiv, \prec)$ where $\prec$ is defined representative-wise: $[u]^{\equiv} \prec [v]^{\equiv}$ iff $u \prec v$.
- Inclusion of sets: $(\mathcal{P}(S), \subseteq)$ is a lattice. It has $\varnothing$ and S as the smallest and greatest element, respectively. $\cup$ and $\cap$ yield the least upper and the greatest lower bound of two sets, respectively.
- The diagonal relation: (A, Δ_A) where $\Delta_A(u, v)$ iff $u = v$ is a order and an equivalence relation.
- Equivalences: For any function $f : A \to B$, the model (A, r) with $r(u, v) = 1$ iff $f(u) = f(v)$ is an equivalence relation (and every equivalence relation is of this form).

Remark 3.43 (Lattices). Lattices occur both in Ex. 3.2 and 3.3 because there are two alternative ways to see them. In a lattice formed from two binary function symbols (which are usually written as $\sqcup$ instead of $\circ$ and $\sqcap$ instead of $*$), we can define the predicate symbol by $x \leq y \leftrightarrow x \sqcap y \doteq x$, which is equivalent to $x \sqcup y \doteq y$. Vice versa, in a lattice formed from a binary predicate symbol, we can define $x \sqcup y$ as the least upper bound and $x \sqcap y$ as the greatest lower bound of x and y.

3.6.3 Negative Examples

It is one of the most unsatisfactory results about FOLEQ that a lot of interesting data types cannot be specified.

Example 3.44. Consider the theory $(\Sigma; \Theta)$ for Peano arithmetic from Ex. 1.34. The **standard natural numbers** are the model

$$SN = (\mathbb{N}, 0, x \mapsto x + 1, +, \cdot)$$

We cannot specify the model class $\{SN\}$, i.e., we cannot give a theory Θ whose only model is $\{SN\}$. This would have be a theory whose theorems are

$$SN^* = \{F \in \mathbf{Sen}(\Sigma) \mid [\![F]\!]^{SN} = 1\}$$

This is no surprise: There are lots of other models that are isomorphic to SN and thus satisfy the same sentences. So the best we can hope for is to specify the class $\{I \in \mathbf{Mod}(\Sigma) \mid I \text{ isomorphic to } SN\}$.

However, not even that works: There are models of Peano arithmetic that are not isomorphic to SN but satisfy exactly the same formulas as SN. The problem is that their difference to SN cannot be expressed in FOLEQ. These are called non-standard models. Consequently, every theory that has SN as a model also has all the non-standard models.

It is hard to visualize non-standard models of arithmetic (see http://en.wikipedia.org/wiki/Non-standard_arithmetic). All of them are total order that start with the natural numbers and are then followed by densely ordered copies of the integers. There are also non-standard models with uncountable universes.

Thus, the best we can do is to axiomatize the class

$$\mathcal{M} = \{I \in \mathbf{Mod}(\Sigma) \mid [\![F]\!]^I = [\![F]\!]^{SN} \text{ for all } F \in \mathbf{Sen}(\Sigma)\}$$

i.e., the class of all models satisfying the same sentences as SN.

But even that is difficult. We can try to axiomatize $\mathcal{M}$ using the axioms Θ of Peano arithmetic. But that is not enough: There are FOLEQ-sentences that hold in the natural numbers, but that are not consequences of Peano arithmetic, e.g., Goodstein's theorem (see http://en.wikipedia.org/wiki/Goodstein%27s_theorem). In particular, Peano arithmetic is not a complete theory.

Essentially the best theory we can find to axiomatize $\mathcal{M}$ is to use SN^* itself as the set of axioms. But that is like giving up: Then all the sentences that we would like to prove are axioms to begin with.

But even that is not a good solution. SN^* is not only infinite but – as we will see in Sect. 5.4 – not even recursively enumerable. Thus, we cannot even write down all the axioms in SN^* or tell which formulas are in it.

However, the theory of Peano arithmetic without multiplication does specify the model class $\{(\mathbb{N}, 0, x \mapsto x + 1, +)\}^{**}$. In particular, Peano arithmetic without multiplication is a complete theory.

Example 3.45. Similarly to Ex. 3.44, the real numbers cannot be specified in FOLEQ.

Example 3.46. Both the natural and the real numbers can be specified in second-order logic (which we have not covered). Second-order logic permits quantifying over subsets of the universe. Using that, we can talk about induction for the natural numbers and, e.g., Dedekind cuts for the real numbers.

Example 3.47. For a given signature,

- the class of all models with a universe of fixed finite size *can* be specified in FOLEQ (Exercise!),
- the class of all models with a universe of fixed infinite size (e.g., countable) *cannot* be specified in FOLEQ,
- the class of all models with a finite universe *cannot* be specified in FOLEQ,
- the class of all models with an infinite universe *can* be specified in FOLEQ if the signature contains a non-nullary function symbol. (Exercise!)

3.7 Model Morphisms

[5]

Model morphisms are an important tool to relate models. They are best understood by example:

Example 3.48 (Monoid Morphisms). A morphism from a monoid $M_1 = (U_1, *_1, e_1)$ to a monoid $M_2 = (U_2, *_2, e_1)$ is a map $\varphi : U_1 \to U_2$ such that

- $\varphi(u *_1 v) = \varphi(u) *_2 \varphi(v)$,
- $\varphi(e_1) = e_2$.

The preserved structure is that of the monoid operations: φ maps composition to composition and unit to unit.

For a more concrete example, consider $M_1 = (\mathbb{N}, +, 0)$ and $M_2 = (\mathbb{Z}, +, 0)$ and let $\varphi : \mathbb{N} \to \mathbb{Z}$ be the inclusion. Clearly, this inclusion is a monoid morphism. More generally, whenever $U_1 \subseteq U_2$ and φ is an inclusion morphism, then we say that M_1 is a submonoid of M_2.

For a less trivial example, consider M_1 and M_2 as before but let now $\varphi(u) = ku$ for some fixed $k \in \mathbb{Z}$. This is a monoid morphism for every k (even for $k = 0$).

For an example that relates very different monoids, consider $M_1 = (\Sigma^*, +, \varepsilon)$ and $M_2 = (\mathbb{N}, +, 0)$ where Σ^* denotes the set of words over some fixed alphabet (i.e., Scala-strings), $+$ denotes the binary function that concatenates two words, and ε denotes the empty word. Then (M_1 is a monoid and) the length of a word is a monoid morphism from M_1 to M_2.

It is easy to prove that the properties of a morphism imply similar laws for all composed expressions of monoids. For example, $\varphi((u *_1 (v *_1 *_1 e_1)) *_1 w) = (\varphi(u) *_2 (\varphi(v) *_2 e_2)) *_2 \varphi(w)$. We say that the morphism **commutes** with the operations. Intuitively, φ can be pulled into (or reversely: out of) the expression. Other words for morphisms are **structural**, **recursive**, or **compositional** translation because the result of applying φ to a composed object is obtained by recursively applying φ to the subobjects and composing the results using the same term structure.

The above examples already indicate how powerful the concept of morphisms is: Relations between very different objects can be expressed succinctly. Furthermore, the definition of morphism does not depend so much on the particular structure that is supposed to be preserved. For monoids, the preserved structure consists of composition and unit.

[5] This section can be skipped.

Chapter 4

Proof Theory

4.1 Introduction

There is a number of philosophical arguments against model theory.

- In order to define consequence, mathematics (i.e., the collection of models) must already exist.
- The semantics of formulas is kind of trivial by interpreting, e.g., $\wedge$ as "and". Are we really formalizing anything, or are we only using fancy symbols?
- The model-theoretic definition of consequence is not accessible to algorithmic treatment.
- Model-theoretic consequence does not correspond to the everyday mathematical practice of proving theorems.
- Gödel's incompleteness shows: Consistency is a big problem.

In the following, we will introduce an inference system that forms the core of *proof theory*.
The most important example of a rule coming up in proof theory is the modus ponens rule

$$\frac{\vdash F \to G \qquad \vdash F}{\vdash G} R_2$$

Here $\vdash F$ is the judgment that F is true.

Further examples are this rule for conjunction and truth:

$$\frac{\vdash F \qquad \vdash G}{\vdash F \wedge G} R_1$$

$$\frac{}{\vdash true} R_3$$

This is the axiom of the so-called excluded middle or tertium non datur:

$$\frac{}{\vdash F \vee \neg F} R_4$$

Then proofs are the derivations of this inference system. For example, a proof of G using hypotheses from $\Delta = \{F_1, F_2, (F_1 \wedge (true \wedge F_2)) \to G\}$ could look like this:

$$\dfrac{\dfrac{\vdash F_1 \qquad \dfrac{\dfrac{}{\vdash true} R_3 \qquad \vdash F_2}{\vdash true \wedge F_2} R_1}{\vdash F_1 \wedge (true \wedge F_2)} R_1 \qquad \vdash (F_1 \wedge (true \wedge F_2)) \to G}{\vdash G} R_2$$

Here we apply R_3 to get $\vdash \mathit{true}$, then R_1 twice to get $\vdash F_1 \wedge (\mathit{true} \wedge F_2)$, then R_2 to get G. Note how the leaves of the derivation tree are either axioms (i.e., $\vdash \mathit{true}$) or hypotheses (i.e., $\vdash F$ for $F \in \Delta$). We can think of the hypotheses as additional axioms.

An inference system used for proof theory is often also called a *calculus*. In fact, first-order logic is often called (first-order) predicate calculus. Calculus-based definitions of consequence have important properties that answer the criticisms against model theory:

- It is algorithmic: Given a calculus, we can
  - search for proofs,
  - check whether a given proof is correct.
- It is defined by giving rules that directly capture our intuition.
- No model theory and almost nothing about mathematics is presupposed.
- It is very similar to how a mathematician would reason.

Now the crucial design question is: How do we get a calculus for PL, FOL, FOLEQ?

- First design decision: What are the judgments?
  - simplest choice: the Σ-judgments are $\vdash F$ for Σ-formulas F
  - more complicated choices of judgments are much more useful in practice
- Second design decision: What are the rules?
  - Very different sets of rules may lead to the same consequence relation
  - Very similar sets of rules may lead to very different implementations
- Conflicting design goals:
  - Theoretical elegance, simplicity
  - Practical strength, efficiency — We want to use calculi to make the computer find proofs!

4.2 Calculi for Propositional Logic

4.2.1 Hilbert-Calculi

The simplest calculi are those where the judgments are simply the formulas. Such calculi are called Hilbert calculi.

Assume a PL-signature Σ. A calculus $C(\Sigma)$ is given by using the set $\mathit{Form}(\Sigma)$ as the set of judgments and the following rules:

- modus ponens
$$\frac{F \to G \quad F}{G}\, MP$$
- the following axioms (i.e.) rules without hypotheses (where we use the $\leftrightarrow$ as an abbreviation)
  1. $\mathit{false} \leftrightarrow \neg \mathit{true}$
  2. $(F \to G) \leftrightarrow (\neg F \vee G)$
  3. $(F \wedge G) \leftrightarrow \neg(\neg F \vee \neg G)$
  4. true
  5. $(F \vee F) \to F$
  6. $F \to (F \vee G)$
  7. $(F \vee G) \to (G \vee F)$
  8. $(G \to H) \to ((F \vee G) \to (F \vee H))$

All these rules are used for arbitrary formulas F and G. Therefore, we actually have infinitely many rules.

This calculus is essentially the one used in the Bertrand's and Russell's Principia [WR13], a very influential work from 1910 that attempted to formalize all of mathematics. For example, Gödel discovered his result [Göd31] while working in the Principia, i.e., they were the mathematics as he knew it.

An overview over Hilbert Calculi for various variants of propositional logics is given at http://home.utah.edu/~nahaj/logic/structures/systems/index.html

Assume $\Sigma = \{p, q\}$. Then an example proof with hypothesis $\Delta = \{p\}$ looks like this:

$$\dfrac{\dfrac{}{(p \vee q) \to (q \vee p)}\ \textit{Axiom 7} \qquad \dfrac{\dfrac{}{p \to (p \vee q)}\ \textit{Axiom 6} \qquad -\Delta}{p \vee q}\ MP}{q \vee p}\ MP$$

Hilbert calculi are extremely useful in theory. Because there is only one rule, the proofs are very simple: The leaves are axioms, and then only modus ponens is used.

For the same reason, Hilbert calculi are extremely useless in practice. For example, try to derive something as trivial as $F \to F$.

4.2.2 Other Calculi

Other calculi for PL are obtained by dropping the rules for the quantifiers from the following calculi for FOL.

4.3 Calculi for First-Order Logic

4.3.1 The Natural Deduction Calculus

The point of natural deduction is that it captures the way in which we reason intuitively and mathematically.

The main judgment of the ND Calculus is

$$\Gamma; \Delta \vdash_\Sigma F$$

where Γ is a Σ-context, Δ is a list of Σ-formulas in context Γ and F is a Σ-formula in context Γ. The rules of the ND calculus are described in detail below.

Notation 4.1. A lot of simplifications are common:

- The subscript Σ in $\vdash_\Sigma$ is often dropped when Σ is fixed.
- We usually write $x_1, \ldots, x_m$ instead of $\{x_1, \ldots, x_m\}$ and Γ, x instead of $\Gamma \cup \{x\}$.
- Similarly, we usually write $F_1, \ldots, F_m$ instead of $\{F_1, \ldots, F_m\}$ and Δ, F instead of $\Delta \cup \{F\}$.
- In $\Gamma; \Delta \vdash F$, we omit Γ or Δ if they are empty.
- In $\Gamma; \Delta \vdash F$, usually Γ is omitted altogether. In that case, it is implied that Γ is the set of all variables occurring free in any formula in Δ or in F.

The intuition of the judgment $\{x_1, \ldots, x_m\}; \{F_1, \ldots, F_n\} \vdash F$ is as follows: For arbitrary values for $x_1, \ldots, x_m$, if all of the $F_1, \ldots, F_n$ hold, then F holds. Using this intuition, the rules of ND capture the meaning we have in mind when we prove something. That's why it is called *natural* deduction.

The rules are split into elimination and introduction rules. We use introduction rules when we establish a proof goal. For example, we use $\wedge I$ to establish the *proof goal* $\vdash A \wedge B$ after first proving $\vdash A$ and $\vdash B$. We use elimination rules when we use assumptions. For example, we use $\wedge E_l$ to break down the previously proved $\vdash A \wedge B$ into $\vdash A$.

When we actually do a proof in this calculus, we typically apply introduction rules backwards (i.e., from bottom to top) as long as possible. Then we use elimination rules forwards (i.e., from top to bottom) to build up the needed proof goal from the assumptions.

Fig. 4.1 gives the introduction and elimination rules for ND as well as some structural rules. All rules are given for arbitrary Γ, Δ, A, and B. To indicate that Σ, Γ, and Δ are irrelevant (and could be omitted), they are printed in gray.

	Introduction	Elimination
$\wedge$	$\dfrac{\Gamma;\Delta \vdash_\Sigma A \quad \Gamma;\Delta \vdash_\Sigma B}{\Gamma;\Delta \vdash_\Sigma A \wedge B}\wedge I$	$\dfrac{\Gamma;\Delta \vdash_\Sigma A \wedge B}{\Gamma;\Delta \vdash_\Sigma A}\wedge E_l \quad \dfrac{\Gamma;\Delta \vdash_\Sigma A \wedge B}{\Gamma;\Delta \vdash_\Sigma B}\wedge E_r$
$\vee$	$\dfrac{\Gamma;\Delta \vdash_\Sigma A}{\Gamma;\Delta \vdash_\Sigma A \vee B}\vee I_l \quad \dfrac{\Gamma;\Delta \vdash_\Sigma B}{\Gamma;\Delta \vdash_\Sigma A \vee B}\vee I_r$	$\dfrac{\Gamma;\Delta \vdash_\Sigma A \vee B \quad \Gamma;\Delta, A \vdash_\Sigma C \quad \Gamma;\Delta, B \vdash_\Sigma C}{\Gamma;\Delta \vdash_\Sigma C}\vee E$
$\rightarrow$	$\dfrac{\Gamma;\Delta, A \vdash_\Sigma B}{\Gamma;\Delta \vdash_\Sigma A \rightarrow B}\rightarrow I$	$\dfrac{\Gamma;\Delta \vdash_\Sigma A \rightarrow B \quad \Gamma;\Delta \vdash_\Sigma A}{\Gamma;\Delta \vdash_\Sigma B}\rightarrow E$
$\neg$	$\dfrac{\Gamma;\Delta, A \vdash_\Sigma \mathit{false}}{\Gamma;\Delta \vdash_\Sigma \neg A}\neg I$	$\dfrac{\Gamma;\Delta \vdash_\Sigma \neg A \quad \Gamma;\Delta \vdash_\Sigma A \quad \Gamma \vdash_\Sigma C : \mathtt{form}}{\Gamma;\Delta \vdash_\Sigma C}\neg E$
true	$\dfrac{}{\Gamma;\Delta \vdash_\Sigma \mathit{true}}\mathit{trueI}$	
false		$\dfrac{\Gamma;\Delta \vdash_\Sigma \mathit{false} \quad \Gamma \vdash_\Sigma C : \mathtt{form}}{\Gamma;\Delta \vdash_\Sigma C}\mathit{falseE}$
$\forall$	$\dfrac{\Gamma, x;\Delta \vdash_\Sigma A \quad x \notin \Gamma}{\Gamma;\Delta \vdash_\Sigma \forall x\ A}\forall I$	$\dfrac{\Gamma;\Delta \vdash_\Sigma \forall x\ A \quad \Gamma \vdash_\Sigma t : \mathtt{term}}{\Gamma;\Delta \vdash_\Sigma A[x/t]}\forall E$
$\exists$	$\dfrac{\Gamma;\Delta \vdash_\Sigma A[x/t] \quad \Gamma \vdash_\Sigma t : \mathtt{term}}{\Gamma;\Delta \vdash_\Sigma \exists x\ A}\exists I$	$\dfrac{\Gamma;\Delta \vdash_\Sigma \exists x\ A \quad \Gamma, x;\Delta, A \vdash_\Sigma C \quad x \notin \Gamma, C}{\Gamma;\Delta \vdash_\Sigma C}\exists E$
	$\dfrac{A \in \Delta}{\Gamma;\Delta \vdash A}\mathit{Axiom} \quad \dfrac{}{\Gamma;\Delta \vdash_\Sigma A \vee \neg A}\mathit{tnd} \quad \dfrac{\Gamma, x;\Delta \vdash_\Sigma A \quad x \notin \Gamma, \Delta, A}{\Gamma;\Delta \vdash_\Sigma A}\mathit{nonempty}$	
	$\dfrac{\Gamma;\Delta \vdash_\Sigma A \quad \Gamma;\Delta, A \vdash_\Sigma B}{\Gamma;\Delta \vdash_\Sigma B}\mathit{cut}$	

Figure 4.1: Natural Deduction Rules

The intuitions behind the rules are as follows.

- $\wedge I$, $\wedge E_l$, $\wedge E_r$: Proving a conjunction is equivalent to proving the two conjuncts.
- $\vee I_l$, $\vee I_r$: To prove a disjunction, it suffices to prove one of the disjuncts.
- $\vee E$: If a disjunction has been proved, it can be used for case distinction on the two disjuncts.
- $\rightarrow I$: Proving an implication means to assume the implicant and to prove the implicate.
- $\rightarrow E$: This is modus ponens.
- $\neg I$: Proving a negation means to prove that the opposite leads to a contradiction. This is also called indirect reasoning. There are various ways to express contradictions; here we use proving $\neg A$ from A; another option is to prove *false* or to prove all formulas (i.e., to prove an arbitrary formula).

- $\neg E$: From a contradiction (i.e., A and $\neg A$ provable), we can prove everything. Compare *false*E.
- *true*I: *true* always holds.
- *false*E: *false* never holds, i.e., if it holds, then there is a contradiction and every well-formed formula holds. Note that there is an additional assumption that C is a well-formed formula. This is necessary because otherwise C could be any nonsensical object.
- $\forall I$: A universal quantification holds, if its body holds for an arbitrary value. The "arbitrary value" is represented by a fresh variable, i.e., a variable that occurs nowhere.
- $\forall E$: If a universal quantification holds, its body can be instantiated with every well-formed term.
- $\exists I$: An existential quantification holds, if its body holds for some value. The "some value" is represented by an arbitrary well-formd term.
- $\exists E$: If an existential quantification holds, then its body must holds for some value. The "some value" is represented by a fresh variable.
- *Axiom*: The assumptions imply themselves.
- *tnd*: Tertium non datur, either a formula or its negation holds.
- *nonempty*: It is permitted to add a fresh variable to the context.
- *cut*: This rule is lemma application, i.e., if a formula is proved separately, then it can be used as an additional assumption.

An important question in a given calculus is the redundancy of rules, especially of the structural rules, which are often not so natural. A rule is redundant if omitting the rule does not change the set of derivable judgments and thus does not change the consequence relation. We have the following.

- The rule *tnd* is not redundant. On the other hand, the system of introduction and elimination rules has a nice symmetry, and adding *tnd* to it seems arbitrary and unnatural. If we omit it, some formulas are not provable anymore. For example, there would be no proofs for $\neg\neg A \to A$ and $\neg\forall x\ F \to \exists x\ \neg F$, which hold in every FOL model. However, some philosophers and researchers argue that would be a feature, not a bug, and that the FOL model theory is wrong, not the proof theory. This point of view is called *intuitionism* and has led to the development of *intuitionistic logic*, which does not use *tnd*.
- The rule *nonempty* is not redundant. It is usually not used in the literature because subtly different and slightly stronger rules for the quantifiers are used. If we omit it, there would be no proof of $\forall x\ A \to \exists x\ A$, which holds in every model with a non-empty universe. Because the FOL model theory assumes that the universe is non-empty we have to add this rule.
- The rule *cut* is redundant. To prove that it is redundant for a given logic, is called *cut* elimination. *cut* elimination is one of the central problems in proof theory because once *cut* is eliminated, it is relatively easy to show consistency. (An inference system is consistent, if not every formula is derivable. An inconsistent inference system is clearly useless.) Also proof search is a lot easier for a machine (For a human, it is harder.) if *cut* is omitted.

Rules for Equality So far we have only considered FOL. By adding the rules from Fig. 4.2, we obtain the natural deduction calculus for FOLEQ.

The intuitions behind the rules are as follows.

- $refl$: Every term is equal to itself (reflexivity).
- *sym*: Equality is symmetric.
- *trans*: Equality is transitive.
- *congterm*: Every function $U(x)$ maps equal inputs s and t to equal outputs $U(s)$ and $U(t)$ (congruence for terms).

$$\frac{\Gamma \vdash_\Sigma t : \mathtt{term}}{\Gamma;\Delta \vdash_\Sigma t \doteq t}\, refl \qquad \frac{\Gamma;\Delta \vdash_\Sigma s \doteq t}{\Gamma;\Delta \vdash_\Sigma t \doteq s}\, sym \qquad \frac{\Gamma;\Delta \vdash_\Sigma r \doteq s \quad \Gamma;\Delta \vdash_\Sigma s \doteq t}{\Gamma;\Delta \vdash_\Sigma r \doteq t}\, trans$$

$$\frac{\Gamma;\Delta \vdash_\Sigma s \doteq t \quad \Gamma, x \vdash_\Sigma U : \mathtt{term}}{\Gamma;\Delta \vdash_\Sigma U[x/s] \doteq U[x/t]}\, congterm$$

$$\frac{\Gamma;\Delta \vdash_\Sigma s \doteq t \quad \Gamma, x \vdash_\Sigma U : \mathtt{form} \quad \Gamma;\Delta \vdash_\Sigma U[x/s]}{\Gamma;\Delta \vdash_\Sigma U[x/t]}\, congform$$

Figure 4.2: Equality Rules

- *congform*: If a property $U(x)$ holds for s then it also holds for anything equal to s (congruence for formulas).

The first three rules have the effect that equality is an equivalence relation. *congterm* makes it additionally a congruence relation. Finally, *congform* say that formulas can never distinguish between equal terms.

The rules *sym* and *trans* are actually redundant: They can be proved using *refl* and *congform* (Exercise!).

The Inference System Finally, we can define the inference system of natural deduction.

Definition 4.2. The inference system $\mathbf{Pf}^{FOLEQ}(\Sigma)$ consists of the following **judgments**

- all judgments $\Gamma \vdash_\Sigma t : \mathtt{term}$ for any Γ, t,
- all judgments $\Gamma \vdash_\Sigma F : \mathtt{form}$ for any Γ, F,
- all judgments $\Gamma;\Theta \vdash_\Sigma F$ for any Γ, Θ, F.

and the proof rules of Fig. 4.1 and 4.2.

The Deduction Theorem Interestingly, we now have two options to define provability:

Definition 4.3. We say that $F \in \mathbf{Sen}^{FOLEQ}(\Sigma)$ is **locally provable** from assumptions $F_1, \ldots, F_n \subseteq \mathbf{Sen}^{FOLEQ}(\Sigma)$ if there is a proof

$$\frac{}{F_1, \ldots, F_n \vdash_\Sigma F}$$

We say that $F \in \mathbf{Sen}^{FOLEQ}(\Sigma)$ is **globally provable** from assumptions $\Theta \subseteq \mathbf{Sen}^{FOLEQ}(\Sigma)$ if there is a proof

$$\frac{\vdash_\Sigma F_1 \quad \ldots \quad \vdash_\Sigma F_n}{\vdash_\Sigma F}$$

Global provability is a very general notion because it can be stated for virtually any calculus, e.g., for a Hilbert calculus. Local provability depends on the structure of the ND calculus: We can only state it because ND judgments may have assumptions, e.g., we cannot state if for a Hilbert calculus.

It turns out for the ND calculus, both notions are equivalent:

Theorem 4.4 (Deduction Theorem)**.** *In the ND calculus for FOLEQ, local and global provability are equivalent.*

Proof. Firstly, assume a provability $F_1, \ldots, F_n \vdash_\Sigma F$. By applying $\to I$ n times, we obtain a proof of $\vdash_\Sigma F_1 \to \ldots \to F_n \to F$. By applying $\to E$ n times, we obtain global provability. The opposite direction proceeds accordingly. □

4.4 An Abstract Definition of the Proof Theory of a Logic

In Sect. 1.4 and 3.4, we have seen abstract definitions of syntax $(\mathbf{Sig}, \mathbf{Sen})$ and for a given logic syntax the model theory $(\mathbf{Mod}, \models)$ of logics. We will now the analogue for proof theory: for a given logic syntax, define the proof theory of a logic.

Definition 4.5 (Proof Theory)**.** A proof theory for the logic syntax $(\mathbf{Sig}, \mathbf{Sen})$ is a pair $(\mathbf{Pf}, \vdash)$ such that for every $\Sigma \in \mathbf{Sig}$, we have that

- $\mathbf{Pf}(\Sigma)$ is an inference system,
- for any sentences $\Theta \subseteq \mathbf{Sen}(\Sigma)$ and $F \in \mathbf{Sen}(\Sigma)$, there is a judgment of $\Theta \vdash F$ in $\mathbf{Pf}(\Sigma)$.

If there is a proof $p \in \mathbf{Pf}(\Sigma)$ of $\Theta \vdash F$ in $\mathbf{Pf}(\Sigma)$, we say F is provable from/implied by/entailed by/a consequence of Θ.

A four-tuple $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Pf}, \vdash)$ of a logic syntax $(\mathbf{Sig}, \mathbf{Sen})$ and a proof theory $(\mathbf{Pf}, \vdash)$ for it is called a *proof theoretical logic*.

Example 4.6. We immediately get some examples:

- We have a proof theoretical logic $FOLEQ = (\mathbf{Sig}^{FOLEQ}, \mathbf{Sen}^{FOLEQ}, \mathbf{Pf}^{FOLEQ}, \vdash^{FOLEQ})$.
  $\mathbf{Pf}^{FOLEQ}$ is defined in Def. 4.2. $\Theta \vdash^{FOLEQ}_{\Sigma} F$ is already defined as an abbreviation for $\varnothing; \Theta \vdash_{\Sigma} F$.
- Proof theoretical logics for FOL, ALGEQ, and HORNEQ are obtained from FOLEQ in the obvious way.

4.5 Theorems and Consequence (Proof-Theoretically)

In Sect. 3.5, we defined model theoretical theorems and consequence for an arbitrary model theory. Now we do the same proof theoretically for an arbitrary proof theory. We obtain the proof theoretical analogues to Def. 3.23, Def. 3.27, Def. 3.30, Lem. 3.31, and 3.32:

Definition 4.7 (Theorems)**.** Given a proof theoretical logic $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Pf}, \vdash)$ and a theory $(\Sigma; \Theta)$. Then a Σ-sentence is a (proof-theoretical) *theorem/tautology/consequence/valid sentence* of $(\Sigma; \Theta)$ if there is a proof of

$$\Theta \vdash_{\Sigma} F$$

and we write

$$Thm(\Sigma; \Theta) = \{F \in \mathbf{Sen}(\Sigma) \mid \Theta \vdash_{\Sigma} F\}$$

F is called a *contradiction* if it $\Theta, F \vdash_{\Sigma} C$ for all $C \in \mathbf{Sen}(\Sigma)$.

Lemma 4.8. *If a proof theoretical logic contains a negation whose rules are as for the ND calculus: F is a theorem iff $\neg F$ is a contradiction. F is a contradiction iff $\neg F$ is a theorem.*

Proof. Clear using the rules $\neg I$ and $\neg E$. □

Definition 4.9. A theory $(\Sigma; \Theta)$ is called

- *inconsistent* if all sentences are theorems, $\Theta \vdash_{\Sigma} F$ for every $F \in \mathbf{Sen}(\Sigma)$,
- *consistent* if it is not inconsistent, i.e., there is a sentence that is not a theorem,
- *complete* if every sentence is either a theorem or a contradiction, i.e., if for every $F \in \mathbf{Sen}(\Sigma)$ either $\Theta \vdash_{\Sigma} F$ or $\Theta \vdash_{\Sigma} \neg F$.

Lemma 4.10. *The following are equivalent for a theory $(\Sigma; \Theta)$ in any proof theoretical logic L:*

1. *It is inconsistent.*

2. *(If L has negation in the same way as FOL:) F and $\neg F$ are theorems for some F.*

3. *(If L has falsity in the same way as FOL:) false is a theorem.*

Proof. Easy using the rules for negation and falsity. □

Notation 4.11. We have defined the notions "consequence", "theorem", "contradiction", "(in)consistent", and "complete" proof-theoretically. All have model-theoretical analogues, see Def. 3.23, 3.27, and 3.30. If we need to distinguish them, we will prefix P, e.g., we will say that a theory is P-consistent. See also Not. 3.33.

Chapter 5

The Relation between Proof and Model Theory

5.1 An Abstract Definition of a Logic

Definition 5.1 (Logic)**.** A *logic* consists of

- a collection of signatures **Sig**,
- a set of sentences $\mathbf{Sen}(\Sigma)$ for every $\Sigma \in \mathbf{Sig}$,
- a collection of models $\mathbf{Mod}(\Sigma)$ for every $\Sigma \in \mathbf{Sig}$,
- a model-theoretic definition of truth and consequence: for every $\Sigma \in \mathbf{Sig}$, a relation $\models_\Sigma$ between $\mathbf{Mod}(\Sigma)$ and $\mathbf{Sen}(\Sigma)$,
- an inference system $\mathbf{Pf}$ (Σ) for every $\Sigma \in \mathbf{Sig}$,
- a proof theoretic definition of consequence: for every $\Sigma \in \mathbf{Sig}$, a judgment $\Delta \vdash_\Sigma F$.

Thus, we have two alternative ways of defining theorems and consequence: model and proof theoretically. The former was strongly influenced by Tarski [TV56] and [Rob50] and has become the dominant method in mathematical logic. The latter was strongly influenced by Hilbert's program [Hil26] and the work by Gödel [Göd30] and Gentzen [Gen34]. It has become important in computer science because it permits using computers to check and search for proofs.

5.2 Soundness and Completeness

Proof and model theory work best when used in combination: We use theories as interfaces, models as interface providers, and the proofs of a theory can be reused in every model. This works if P-consequence and M-consequence are the same, which is where soundness and completeness come in.

5.2.1 Definitions

We define:

Definition 5.2 (Soundness and Completeness)**.** A *logic* is *sound* if for all signatures Σ, all sets of sentences Δ and all sentences F,

$$\Delta \vdash_\Sigma F \quad \text{implies} \quad \Delta \models_\Sigma F.$$

A *logic* is *complete* if for all signatures Σ, all sets of sentences Δ and all sentences F,

$$\Delta \models_\Sigma F \quad \text{implies} \quad \Delta \vdash_\Sigma F.$$

In other words, it is complete if M-consequence implies P-consequence; or if all M-theorems are P-theorems. And it is sound if the implication goes the other way around.

Notation 5.3. Note that both theories and logics can be complete. The two notions are only loosely related and should be seen as different notions that happen to share the name "complete".

Soundness and completeness extend to the other notions mentioned in Not. 3.33 and 4.11:

Theorem 5.4. *If a logic is sound then,*

- *P-contradictions are M-contradictions,*
- *P-inconsistent theories are M-inconsistent,*
- *P-complete theories are M-complete.*

For completeness, the opposite implications hold.

Proof. Straightforward because all notions are defined in terms of the consequence relation. □

Note that Thm. 5.4 does not include a case for "consistent" theories. For "consistent", the implication is flipped because "M-consistent theories are P-consistent" is the same as "not M-inconsistent theories are not P-inconsistent" and that is equivalent to "P-inconsistent theories are M-inconsistent". More precisely, we have the following:

5.2.2 Intuitions

Soundness means that the proof theory is correct: If we can prove a consequence, it holds about all models. It is usually easy to show soundness by induction on proof trees.

Completeness is more complicated (and there are useful logics that are not complete). To understand it better, we show the following:

Lemma 5.5. *The following are equivalent for any logic L that has a negation together with rules for negation introduction and double-negation elimination (i.e., $\neg\neg F \vdash F$):*

1. *L is complete (i.e., M-consequence implies P-consequence).*
2. *Every P-consistent theory has a model (i.e., P-consistency implies M-consistency).*

Proof. By circular implication:
1 implies 2: See Thm. 5.4. 2 implies 1: To show completeness, assume a theory $(\Sigma;\Theta)$ and an M-theorem F (*). Then $(\Sigma;\Theta\cup\{\neg F\})$ has no model. Using 2, we obtain that $(\Sigma;\Theta\cup\{\neg F\})$ is P-inconsistent. Using negation introduction, we can derive $\neg\neg F$ from Θ, and because of double-negation elimination, we can also derive F. Thus, F is a P-theorem of $(\Sigma;\Theta)$. □

Lem. 5.5 is the typical way how we prove completeness.

Then we have two intuitions for the word "completeness". If the model theory defines the theorems, completeness is a property of the proof theory: A logic is complete if it has enough proofs to derive all the theorems.

If the proof theory defines the theorems, completeness is a property of the model theory: A logic is complete if it has enough models to interpret all the consistent theories.

Another way to think about soundness and completeness is to imagine a step-by-step logic design. Say initially we have no proof rules and no models. Then no formula is P-theorem, and all formulas are M-theorems. The logic is as sound as it can be but also as far from complete as it can be.

We can change the logic by adding proof rules (which increases the set of P-theorems) and/or by adding models (which decreases the set of M-theorems). If we have added enough proof rules and models, the sets of P-theorems and M-theorems meet, and we have a sound and complete logic. If we keep adding proof rules or models, we will have too many: The set of P-theorems becomes bigger than the set of M-theorems, and we lose soundness (while staying complete).

5.2.3 Compactness

Compactness is the property that all consequences are caused by finitely many axioms:

Definition 5.6. A model theory is called **compact** if $\Sigma \models_\Theta F$ implies $\Sigma \models_{\Theta'} F$ for some finite set $\Theta' \subseteq \Theta$.

Theorem 5.7. *If a logic is sound and complete, then it is compact.*

Proof. We always have that $\Theta \vdash_\Sigma F$ implies $\Theta' \vdash_\Sigma F$ for some finite set $\Theta' \subseteq \Theta$ because a proof can only use finitely many axioms from Θ.

Therefore, soundness and completeness together imply compactness. □

5.3 The Big Theorems of First-Order Logic

The most important theorems about FOL and FOLEQ are soundness and completeness. The first sound and complete calculi for first-order logic were given by Gödel [Göd30] and Gentzen [Gen34] in the 1930s.

5.3.1 Soundness

Theorem 5.8 (Soundness of FOL)**.** *FOL is sound.*

Proof. By induction on derivations. The theorem only mentions those judgments where $\Gamma = \varnothing$ (i.e., Γ is omitted). This is as usual the only case we are really interested in. But in the induction we will encounter all judgments.

Therefore, we have to prove something more general, namely:

$\Gamma; \Delta \vdash_\Sigma F$ implies that
for every model $I \in \mathbf{Mod}(\Sigma)$ and every assignment α for Γ,
we have that
if $I, \alpha \models_\Sigma T$ for all $T \in \Delta$,
then $I, \alpha \models_\Sigma F$.

Note that if we put $\Gamma = \varnothing$, this is the same as saying $\Delta \models_\Sigma F$.

Now we can prove this more general statement by induction on derivations. The intuition behind this is the following: For all axioms $\frac{}{\vdash A}$, show that A is true in every model. For every rule $\frac{J_1 \quad \dots \quad J_n}{J}$ show that applying the rule preserves truth, i.e., show the above for J assuming it is true (induction hypothesis) for the J_i. Since proof trees are built up by composing rules, the induction requires one case for every rule. Therefore, we often say that a single rule is sound: The soundness of a rule R means that the induction step succeeds for R; then to show soundness of the logic, we need to show that every rule is sound.

In the following we give three example soundness proofs for the rules *Axiom*, $\wedge I$, and $\to I$.

The case for *Axiom*. For this rule, there are no hypotheses. So we have to show that for every model and assignment I and α, if $I, \alpha \models_\Sigma T$ for all $T \in \Delta$, then $I, \alpha \models_\Sigma A$. This is trivial because $A \in \Delta$.

The case for $\wedge I$. For this rule, there are two hypotheses. Applying the induction hypothesis to them yields:

- (1) For every model and assignment I and α, if $I, \alpha \models_\Sigma T$ for all $T \in \Delta$, then $I, \alpha \models_\Sigma A$.
- (2) For every model and assignment I and α, if $I, \alpha \models_\Sigma T$ for all $T \in \Delta$, then $I, \alpha \models_\Sigma B$.

We have to show: For every model and assignment I and α, if $I, \alpha \models_\Sigma T$ for all $T \in \Delta$, then $I, \alpha \models_\Sigma A \wedge B$. This is obvious. But to be totally clear, here is the full proof:

- We pick an arbitrary model I_0 and an arbitrary assignment α_0.
- We assume that (3) $I_0, \alpha_0 \models_\Sigma T$ for all $T \in \Delta$. We have to show $I_0, \alpha_0 \models_\Sigma A \wedge B$.
  - From (3), using (1) and (2), we obtain (4) $I_0, \alpha_0 \models_\Sigma A$ and (5) $I_0, \alpha_0 \models_\Sigma B$, respectively.

  - From (4) and (5), using the definition of the interpretation of formulas, we obtain $I_0, \alpha_0 \models_\Sigma A \wedge B$.
  - Done.
- Done.

The case for $\to I$. For this rule, there is one hypothesis. Applying the induction hypothesis to it yields:

- (1) For every model and assignment I and α, if $I, \alpha \models_\Sigma T$ for all $T \in \Delta \cup \{A\}$, then $I, \alpha \models_\Sigma B$.

We have to show: For every model and assignment I and α, if $I, \alpha \models_\Sigma T$ for all $T \in \Delta$, then $I, \alpha \models_\Sigma A \to B$.

- We pick an arbitrary model I_0 and an arbitrary assignment α_0.
- We assume that (2) $I_0, \alpha_0 \models_\Sigma T$ for all $T \in \Delta$. We have to show $I_0, \alpha_0 \models_\Sigma A \to B$.
- We distinguish two cases: $I_0, \alpha_0 \models_\Sigma A$ and $I_0, \alpha_0 \not\models_\Sigma A$.
  - $I_0, \alpha_0 \not\models_\Sigma A$. Then by definition of the interpretation of formulas $I_0, \alpha_0 \not\models_\Sigma A \to B$. Done.
  - (3) $I_0, \alpha_0 \models_\Sigma A$.
    * Combining (2) and (3), we are able to apply (1), from which we obtain (4) $I_0, \alpha_0 \models_\Sigma B$.
    * From (3) and (4), using the definition of the interpretation of formulas, we obtain $I_0, \alpha_0 \models_\Sigma A \to B$.
    * Done.
- Done.

□

5.3.2 Completeness

The completeness of FOL requires a few additional constructions.

The central idea of the completeness proof is to build a model S of a theory $(\Sigma; \Theta)$ from the terms and proofs. First, we define the interpretation of function symbols – this is called the term model:

Definition 5.9 (Term Model)**.** Let Σ be a signature without predicate symbols. If it does not have a nullary function symbol, we add one.

Then we define the model S as follows:

- $\texttt{term}^S = \{t \mid \vdash_\Sigma t : \texttt{term}\}$
- $f^S(u_1, \ldots, u_n) = f(u_1, \ldots, u_n)$ for $n = ar(f)$

The nullary function symbol is only needed to make sure that $\texttt{term}^S$ is not empty.

Read the last line in Def. 5.9 very carefully:

1. $\texttt{term}^S$ is a set, which S defines to be set of terms.
2. f is a function symbol.
3. f^S is an n-ary function on $\texttt{term}^S$, i.e., a function that takes terms and returns a term.
4. $f^S(u_1, \ldots, u_n)$ is the application of f^S to some arguments, which in this case must be terms.
5. $f(u_1, \ldots, u_n)$ is the term formed from the function symbol f and the term u_i.

The important property of the term model is that it interprets every term as itself:

Lemma 5.10. *In the situation of Def. 5.9, we have*

$$[\![t]\!]^S = t$$

for all terms $\vdash_\Sigma t : \texttt{term}$.

Proof. We use induction on t:

- Case $t = f(t_1, \ldots, t_n)$: We have

$$[\![f(t_1, \ldots, t_n)]\!]^S =^{Def} f^S([\![t_1]\!]^S, \ldots, [\![t_n]\!]^S) =^{IH} f^S(t_1, \ldots, t_n) =^{Def} f(t_1, \ldots, t_n)$$

- Case $t = x$: impossible because t has no free variables.

□

You should read the proof very carefully, too, and understand how the three steps in the case for function terms proceed.

We cannot interpret every formula as itself because the set $\texttt{form}^S$ is already fixed to be $\{0, 1\}$. But we can do something similar: We interpret a formula F as 1 if it is a theorem and as 0 if is a contradiction. But that does not quite work immediately: If Θ is not complete, there are formulas that are neither theorems nor contradictions. Therefore, we show:

Lemma 5.11. *For every P-consistent theory Θ, there is a P-consistent complete theory $\overline{\Theta}$ with $\Theta \subseteq \overline{\Theta}$.*

Proof. Let $F_1, F_2, \ldots,$ be an enumeration of all Σ-sentences (in any order). We define theories $\Theta_0, \Theta_1, \Theta_2, \ldots$ inductively:

- $\Theta_0 = \Theta$
- $\Theta_{n+1} = \begin{cases} (a) & \Theta_n \cup \{F_{n+1}\} \quad \text{if not } \Theta_n \vdash_\Sigma \neg F_{n+1} \\ (b) & \Theta_n \quad \text{otherwise} \end{cases}$

We define $\overline{\Theta} = \bigcup_{i \in \mathbb{N}} \Theta_i$.

We have to show three properties about $\overline{\Theta}$.

- $\Theta \subseteq \overline{\Theta}$: That is obvious.
- Completeness: Consider a sentence F. We know $F = F_n$ for some n. By construction $\overline{\Theta}$ contains F_n for every sentence or can prove $\neg F_{n+1}$. Thus it can prove F_n or $\neg F_n$.
- P-consistency:
  1. We show that if Θ_{n+1} is P-inconsistent, then so is Θ_n. Assume Θ_{n+1} is P-inconsistent. We distinguish the two cases in the definition of Θ_{n+1}:
     (a) Then we have $\Theta_n, F_{n+1} \vdash_\Sigma \mathit{false}$ and thus (using the negation introduction rule) $\Theta \vdash_\Sigma \neg F_{n+1}$. But that violates the assumption of case (a). There are two cases for G:
     (b) Then $\Theta_{n+1} = \Theta_n$, so Θ_n is also inconsistent.
  2. Thus, because Θ_0 is P-consistent, so are all Θ_n.
  3. If $\overline{\Theta}$ were P-inconsistent, there would be a proof p of $\overline{\Theta} \vdash_\Sigma \mathit{false}$. Because a proof can only use finitely many axioms, there would be a Θ_n that contains all axioms used in p. Then Θ_n would be inconsistent, too, which is impossible.

□

Then we can extend Def. 5.9:

Definition 5.12 (Term Model with Predicates)**.** For a theory $(\Sigma; \Theta)$, we define the model S as follows:

- $\texttt{term}^S$ and f^S are as in Def. 5.9.
- $p^S(u_1, \ldots, u_n) = \begin{cases} 1 & \text{if } \Theta \vdash_\Sigma p(u_1, \ldots, u_n) \\ 0 & \text{otherwise} \end{cases}$

Correspondingly, we can extend Def. 5.10:

Lemma 5.13. *In the situation of Def. 5.12 let Θ be complete and consistent. Then, we have*

$$[\![F]\!]^S = \begin{cases} 1 & \text{if } \Theta \vdash_\Sigma F \\ 0 & \text{if } \Theta \vdash_\Sigma \neg F \end{cases}$$

for all sentences $\vdash_\Sigma F : \texttt{form}$ that do not use $\forall$, $\exists$, or $\doteq$.

Proof. We use induction on F:

- Case $F = p(t_1, \ldots, t_n)$: We have

$$[\![p(t_1, \ldots, t_n)]\!]^S =^{Def} p^S([\![t_1]\!]^S, \ldots, [\![t_n]\!]^S) =^{\text{Lem. 5.10}} p^S(t_1, \ldots, t_n) =^{Def} \begin{cases} 1 & \text{if } \Theta \vdash_\Sigma p(u_1, \ldots, u_n) \\ 0 & \text{if } \Theta \vdash_\Sigma \neg p(u_1, \ldots, u_n) \end{cases}$$

  Here, the last step uses the fact that Θ is complete to turn "otherwise" into "if $\Theta \vdash_\Sigma \neg p(u_1, \ldots, u_n)$".

- Case $F = G \wedge H$: We distinguish two cases:
  - $[\![G]\!]^S = 1$ and $[\![H]\!]^S = 1$: We evaluate both sides of the equation:
    (left) We get $[\![G \wedge H]\!]^S = 1$.
    (right) The induction hypothesis for G and H yields $\Theta \vdash_\Sigma G$ and $\Theta \vdash_\Sigma H$. Thus, (using the conjunction introduction rule), we have $\Theta \vdash_\Sigma G \wedge H$. Thus, the right side also yields 1.
  - $[\![G]\!]^S = 0$ or $[\![H]\!]^S = 0$: We evaluate both sides of the equation:
    (left) We get $[\![G \wedge H]\!]^S = 0$.
    (right) The induction hypothesis for G or H yields $\Theta \vdash_\Sigma \neg G$ or $\Theta \vdash_\Sigma \neg H$. Thus, (using the conjunction elimination rules and negation introduction+elimination rules), we have $\Theta \vdash_\Sigma \neg(G \wedge H)$. Thus, the right side also yields 0.

- The cases for negation, disjunction, and implication proceed like the one for conjunction.

□

Remark 5.14. There is a more complex version of Lem. 5.13 that applies to all sentences. However, all of the above definitions have to be changed to use a different set $\texttt{term}^S$. The key ideas is

- To handle equality, $\texttt{term}^S$ is the quotient of the set of terms modulo the equivalence relation $\Theta \vdash_\Sigma t_1 \doteq t_2$. This is necessary to make sure $[\![t_1]\!]^S = [\![t_2]\!]^S = 1$ whenever $\Theta \vdash_\Sigma t_1 \doteq t_2$.
- To handle quantifiers, Σ and Θ have to be extended: If we can prove $\forall x_1 \ldots \forall x_n\ \exists x A$, we have to add an n-ary function symbols f and an axiom $\forall x_1 \ldots \forall x_n\ A[x/f(x_1, \ldots, x_n)$. This is necessary to make sure that $\texttt{term}^S$ contains the element necessary for $[\![\exists x\ A]\!]^{=}1$.

Finally, we can prove completeness of FOL:

Theorem 5.15 (Completeness of FOL)**.** *FOL is complete.*

Proof. We only prove the completeness of FOL with $\forall$, $\exists$, and $\doteq$ removed. The general prove proceeds similarly after generalizing Lem. 5.13.

By Lem. 5.5, we only have to show that every P-consistent FOL theory has a model. So assume a P-consistent theory $(\Sigma; \Theta)$.

Using Lem. 5.11, we can assume that Θ is complete. Otherwise, we construct a model of $\overline{\Theta}$, which will also be a model of Θ.

We use the term model from Def. 5.12. By Lem. 5.13, it satisfies all formulas in Θ. □

5.4 Incompleteness

In Ex. 3.44, we already saw that we cannot specify the natural numbers using Peano arithmetic in FOLEQ. We can now generalize this result.

Theorem 5.16. *The set SN^* from Ex. 3.44 is not recursively enumerable.*

Proof. Omitted. □

We immediately get the following corollary with far-reaching consequences:

Theorem 5.17 (Gödel's First Incompleteness Theorem)**.** *There is no logic in which there is a P-consistent theory whose*

- *signature includes the symbols of Peano arithmetic,*
- *whose theorems include SN^*,*
- *whose axioms are recursively enumerable.*

Proof. If there were such a logic and such a theory, we could recursively enumerate all its theorems and thus SN^*. □

This is one of the most famous results of computer science. It was discovered by Gödel in 1930 [Göd31]. At around that time mathematicians were driven by Hilbert's program [Hil26] to find a logic that can prove all theorems of mathematics. Gödel's incompleteness result showed that is not even possible to find a theory that can prove all theorems about the natural numbers (unless the theory is inconsistent, of course). Since many data types somehow contain the natural numbers, it implies a lot of negative results for other data types.

We can understand this as a special case of the philosophical map-territory problem: A map is different from the mapped territory. In some sense, the best map would have scale 1 : 1 and be an exact copy of the territory; but that is impossible or at least impractical. Similarly, a theory is different from the intended model, and using an exact copy of the model would be impossible or impractical.

Gödel's result also imply a second incompleteness theorem: No matter what logic and what theory we use for mathematics, we will never be able to prove that the theory is consistent. In other words, no language can prove its own consistency. Consistency must be always be proved in a stronger language. But then we do not know whether that stronger language is consistent, and so on.

Therefore, it is theoretically possible that tomorrow someone will discover that set theory (see Ex. 1.30) is inconsistent. And then all of mathematics would break down. This is essentially what happened in 1897 and 1901, when Peano and Russell found inconsistencies in the theories used for mathematics. These theories had been mainly formalized by Frege in 1879 [Fre79] and Cantor in 1883 [Can83] and had been used informally by all mathematicians before.

This gave rise to the development of modern set theories whose axioms are chosen very carefully to make inconsistency unlikely. The currently used variants of set theory have been developed mainly between 1900 and 1930, but mathematicians are still contributing important details. Parallel to the introduction of these new theories, researchers started studying logic and consistency in more detail. Then Gödel's result crushed all hope to settle the consistency question permanently.

Chapter 6

Induction

The prototypical example of induction is mathematical induction, i.e., induction on the natural numbers. In computer science, induction is a more general principle about "doing something exactly once for all elements of a given set A". The "something" can be either of two things: a definition or a proof.

In a definition by induction, we define the function value $f(a)$ for all $a \in A$ by case distinction on a. Then we use the induction principle to argue that f is well-defined, i.e., that each $a \in A$ is covered by exactly one case.

In a proof by induction, we prove the property $P(a)$ for all $a \in A$ by case distinction on a. Then we use the induction principle to argue that P is proved, i.e., that each element of A is covered by exactly one case.

6.1 Mathematical Induction

The induction principle for the natural numbers $\mathbb{N}$ rests on the following:

Definition 6.1 (Natural Numbers)**.** $\mathbb{N}$ is defined as follows:

1. $0 \in \mathbb{N}$
2. if $n \in \mathbb{N}$, then $s(n) \in \mathbb{N}$ (We call $s(n)$ the successor of n, e.g., $s(0) = 1$.)
3. The natural numbers are exactly the ones constructed by the above two cases, i.e.,
   (a) All objects obtained by the cases are different: $0 \neq s(n)$ for any n, and $s(n) \neq s(n')$ for any $n \neq n'$.
   (b) Every $n \in \mathbb{N}$ is obtained by one the cases: for every $n \in \mathbb{N}$, we have $n = 0$ or $n = s(n')$ for some $n' \in \mathbb{N}$.

Inductive Definition Consequently, we can define a function $f \in B^{\mathbb{N}}$ by giving $f(n) \in B$ by induction on $n \in \mathbb{N}$. We have to give two cases

- a case $f(0) \in B$,
- a case $f(s(n))$ where we may use the value $f(n) \in B$.

More formally, we have:

Theorem 6.2 (Definition by Induction)**.** *Given a value $V_0 \in B$ and a function $V_s \in B^B$, the function $f \in B^{\mathbb{N}}$ given by*

$$f(0) = V_0$$
$$f(s(n)) = V_s(f(n))$$

is well-defined.

Example 6.3 (Addition). We define $m + n$ by induction on m.

Technically, this means we define a function f from $\mathbb{N}$ to $B = \mathbb{N}^{\mathbb{N}}$. f is the curried version of addition, i.e., we put $m + n = f(m)(n)$.

The cases for f are

$$\begin{aligned} f(0) &= n \mapsto n \\ f(s(m)) &= n \mapsto s(f(m)(n)) \end{aligned}$$

A more intuitive way to write these cases is:

$$f(0)(n) = 0 + n = n$$

$$f(s(m))(n) = s(m) + n = s(m + n)$$

Example 6.4 (Multiplication). We define $m \cdot n$ by induction on m.

Left as an exercise.

Inductive Proof Similarly, we can prove a property P by proving $P(n)$ by induction on $n \in \mathbb{N}$. We have to give two cases

- a proof of $P(0)$,
- a proof of $P(s(n))$ where we may assume that $P(n)$ holds.

More formally, we have:

Theorem 6.5 (Proof by Induction)**.** *Assume $P(0)$ and "for all $n \in \mathbb{N}$, if $P(n)$ then $P(s(n))$". Then $P(n)$ for all $n \in \mathbb{N}$.*

Example 6.6 (Addition from the Right). We prove $m + 0 = m$ (AddZeroRight) by induction on m:

- Case $m = 0$. We have to prove $0 + 0 = 0$, which holds by applying the definition of $+$.
- Case $m = s(m')$. We have to prove that if $m' + 0 = m'$ (IH), then $s(m') + 0 = s(m')$. We prove this by

  $$s(m') + 0 =^{\text{def}} s(m' + 0) =^{IH} s(m')$$

  where def refers to the definition of $+$.

Similarly, we can show that $m + s(n) = s(m + n)$ (AddSuccRight) by induction on m:

- Case $m = 0$. We have to prove $0 + s(n) = s(0 + n)$, which holds by applying the definition of $+$ twice.
- Case $m = s(m')$. We have to prove that if $m' + s(n) = s(m' + n)$ (IH), then $s(m') + s(n) = s(s(m') + n)$. We prove this by

  $$s(m') + s(n) =^{\text{def}} s(m' + s(n)) =^{IH} s(s(m' + n))$$

  and

  $$s(s(m') + n) =^{\text{def}} s(s(m' + n))$$

Example 6.7 (Commutativity of Addition). We prove commutativity of addition $m + n = n + m$ by induction on m. Left as an exercise.

Example 6.8 (Associativity of Addition). We prove associativity of addition $l + (m + n) = (l + m) + n$ by induction on m.

- Case $m = 0$. We have to prove $l + (0 + n) = (l + 0) + n$. We prove this by

  $$l + (0 + n) =^{\text{def}} l + n$$

and

$$(l + 0) + n =^{\text{AddSuccRight}} l + n$$

- Case $m = s(m')$. We have to prove that if $l+(m'+n) = (l+m')+n$ (IH) then $l+(s(m')+n) = (l+s(m'))+n$. We prove it as follows:

$$l + (s(m') + n) =^{\text{def}} l + s(m' + n) =^{\text{AddSuccRight}} s(l + (m' + n))$$

and

$$(l + s(m')) + n =^{\text{AddSuccRight}} s(l + m') + n =^{\text{def}} s((l + m') + n)$$

and these two are equal due to (IH).

Example 6.9 (Multiplication from the Right). We prove $m \cdot 0 = 0$ (MultZeroRight) by induction on m.
Similarly, we can show that $m \cdot s(n) = m \cdot n + m$ (MultSuccRight) by induction on m.
Left as an exercise.

Example 6.10 (Commutativity of Multiplication). We prove commutativity of addition $m \cdot n = n \cdot m$ by induction on m.
Left as an exercise.

Example 6.11 (Distributivity of Multiplication over Addition). We prove left-distributivity $l \cdot (m+n) = l \cdot m+l \cdot n$ by induction on l.
Left as an exercise.
Then we immediately have right-distributivity $(m+n)\cdot l = m\cdot l+n\cdot l$ using left-distributivity and commutativity.

Example 6.12 (Associativity of Multiplication). We prove associativity of multiplication $l \cdot (m \cdot n) = (l \cdot m) \cdot n$ by induction on m.
Left as an exercise.

Example 6.13 (Neutral Element of Multiplication). We put $1 = s(0)$ and prove $1 \cdot m = m$ and $m \cdot 1 = m$ (no induction necessary).
Left as an exercise.

6.2 Regular Induction

Given an alphabet Σ, we have the following induction principle for the set Σ^*:

Definition 6.14. Σ^* is defined by

- $\varepsilon \in \Sigma^*$,
- for every $x \in \Sigma$: if $w \in \Sigma^*$, then $xw \in \Sigma^*$,
- The elements of Σ^* are exactly the objects obtained by the above cases.

Note that the number of cases is $|\Sigma| + 1$.

Inductive Definition We give some examples for inductive definitions:

Example 6.15 (Reversal). We define the reversal w^R by induction on w

$$\varepsilon^R = \varepsilon$$

$$(xw)^R = w^R x$$

Example 6.16 (Length). We define the length $|w|$ by induction on w

$$|\varepsilon| = 0$$

$$|xw| = s(|w|)$$

Example 6.17 (DFA Transition Function). Given a DFA with transition function $\delta(q, x) \in Q$, we define $\delta^*(q, w)$ by induction on w:

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, xw) = \delta^*(\delta(q, x), w)$$

Inductive Proof We give some examples for inductive proofs:

Example 6.18 (Reversal with Symbols on the Right). We prove $(wy)^R = yw^R$ for $y \in \Sigma$ by induction on w.

- Case $w = \varepsilon$: We have to prove $(\varepsilon y)^R = y\varepsilon^R$. This holds because both sides are equal to y.
- Case $w = xw'$. We have to prove that if $(w'y)^R = yw'^R$, then $(xw'y)^R = y(xw')^R$. We prove it as follows:

$$(xw'y)^R =^{\text{def}} (w'y)^R x =^{IH} yw'^R x$$

and

$$y(xw')^R =^{\text{def}} yw'^R x$$

Example 6.19 (Length with Symbols on the Right). We prove $|wx| = s(|w|)$ (LengthRight) by induction on w.
Left as an exercise.

Example 6.20 (Length-Preservation of Reversal). We prove $|w^R| = |w|$ by induction on w.
Left as an exercise.

Example 6.21 (Self-Inverseness of Reversal). We prove $(w^R)^R = w$ by induction on w.

- Case $w = \varepsilon$: We have to prove $(\varepsilon^R)^R = \varepsilon$, which we obtain by applying the definition of reversal twice.
- Case $w = xw'$. We have to prove that if $(w'^R)^R = w'$ (IH), then $((xw')^R)^R = xw'$. This is left as an exercise.

Regular Induction as a Special Case of Mathematical Induction Computer science prefers using induction on Σ^* as a stand-alone principle.

Mathematics prefers reducing it to induction on natural numbers. Then induction on the words $w \in \Sigma^*$ becomes induction on the length $l = |w|$ of w. The case $l = 0$ corresponds to the case $w = \varepsilon$. And the case $l = s(l')$ corresponds to the case $w = xw'$.

6.3 Context-Free Induction

Given an unambiguous context-free grammar (V, Σ, P, S), the context-free induction principle rests on the following:

Definition 6.22 (Produced Language)**.** For every $A \in V$, the set of words $L(A) \subseteq \Sigma^*$ is defined by

- For every production $A \to w_0 A_1 w_1 \dots w_{n-1} A_n w_n \in P$ (where $A_i \in V$ and $w_i \in \Sigma^*$): if $v_i \in L(A_i)$, then $w_0 v_1 w_1 \dots w_{n-1} v_n w_n \in L(A)$,
- The words in $L(A)$ are exactly the ones obtained by the above cases.

Remark 6.23. Note that the context-free induction principle defines finitely many sets at the same time: $L(A)$ for every $A \in V$. Typically we are only interested in $L(S)$ where S is the start symbol. But it is not possible to define only $L(S)$ – the definition of $L(S)$ must refer to $L(A)$ for other non-terminals A. Therefore, we define $L(A)$ for all $A \in V$ together. We speak of *mutual induction*.

Remark 6.24. If the grammar is ambiguous, then we cannot say "exactly" in Def. 6.22. The cases still cover all words, but some words are covered multiple times. This is harmless for proofs by induction (proving something twice is harmless) but illegal for definition by induction (defining something twice is only allowed if both definitions are the same).

Therefore, we require an unambiguous grammar.

Inductive Definition Given a function

$$f \in \mathcal{SET}^V$$

which maps every non-terminal symbol to a set, we define a set of functions $f_A : f(A)^{L(A)}$, i.e., one function for every non-terminal symbol A. Each function f_A maps the words $w \in L(A)$ produced from A to elements of the set $f(A)$.

The cases of a mutually inductive definition of the f_A are

- for every production $A \to w_0 A_1 w_1 \dots w_{n-1} A_n w_n \in P$: a value $f_A(w_0 v_1 w_1 \dots w_{n-1} v_n w_n) \in f(A)$ where we may use the values $f_{A_i}(v_i) \in f(A_i)$.

Example 6.25 (Semantics of Propositional Logic). Def. 3.2 is a simple example. Here `form` is the only non-terminal symbol, so we only define one function $[\![-]\!] = f_{\texttt{form}}$ from $L(\texttt{form})$ to $f(\texttt{form}) = \{0, 1\}$.

Example 6.26 (Free Variables). Def. 1.13 uses context-free induction to define FV. There are two non-terminals and both $FV(\texttt{term}) = FV(\texttt{form}) = \mathcal{P}(\text{Var})$ where Var is the set of possible variable names. Then Def. 1.13 gives one case per production.

Example 6.27 (Serialization). The serialization of Bonus Exercise 1 is a typical example.

Here for every $A \in V$, $f(A)$ is the set of strings. The functions f_A map every word $w \in L(A)$ to its string representation.

Example 6.28 (Regular Induction). If we specialize to a right-linear grammar and require $f(A)$ to be the same set for all $A \in V$, we obtain regular induction as a special case.

Example 6.29 (Mathematical Induction). We obtain mathematical induction on the natural numbers as the special case of context-free induction on the grammar

$$N \quad ::= \quad 0 \quad | \quad succ(N)$$

Inductive Proofs Inductive proofs proceed very similarly. We do not give details here. However, keep in mind that all examples from Sect. 6.1 are examples of context-free induction via Ex. 6.29.

Context-Free Induction as a Special Case of Mathematical Induction Like regular induction, we can reduce context-free induction to induction on the natural numbers. The key idea is to use induction on the length of the derivation of a word. Alternatively, we can use induction on the height (= length of the longest branch) of the parse tree.

6.4 Context-Sensitive Induction

Context-sensitive induction is a generalization of context-free induction. Instead of functions $f_A \in f(A)^{L(A)}$, we use functions $f_A \in f(A)^{\text{Cont} \times L(A)}$ where Cont is the context.

The context may be different for every inductive definition. Typically, it is a list of identifiers that have been declared together with additional information about every identifier. The context changes during the induction; in particular, declarations are added to the context when they are encountered.

Remark 6.30. Usually, we are only interested in the function $f_S(\varnothing, w)$ where S is the start symbol and $\varnothing$ the empty context. But just like context-free induction requires a function for every non-terminal, context-sensitive induction additionally requires taking an arbitrary context as an argument.

Inductive Definition Context-sensitive inductive definitions abound in theoretical computer science. Many non-trivial operations – e.g., compiling a program – are technically solved in this way (even if they are not called that way).

Example 6.31 (Substitution). Def. 1.22 uses context-sensitive induction to define substitution application.

The induction context is a FOL-substitution γ, which stores which variables have been declared and which term each variable is substituted with.

$f(\texttt{term})$ is the set of terms, and $f(\texttt{form})$ is the set of formulas. The functions $f_{\texttt{term}}(\gamma, t)$ and $f_{\texttt{form}}(\gamma, F)$ are written $\overline{\gamma}(t)$ and $\overline{\gamma}(F)$.

Def. 1.22 gives one case per production. Note how the context is used in (and only in) the case $\overline{\gamma}(x)$.

Inference systems are the prototypical example of context-sensitive induction:

Example 6.32 (Well-Formed Syntax). Def. 2.5 uses context-sensitive induction to define the well-formed terms and formulas for a fixed signature Σ.

The induction context is a FOL-context Γ, which stores which variables have been declared.

The sets $f(\texttt{term})$ and $f(\texttt{form})$ contain boolean truth values, i.e., both $f_{\texttt{term}}$ and $f_{\texttt{form}}$ return either "holds" or "does not hold". We write $\Gamma \vdash_\Sigma t : \texttt{term}$ or $\Gamma \vdash_\Sigma F : \texttt{form}$ if $f_{\texttt{term}}(\Gamma, t)$ holds or $f_{\texttt{form}}(\Gamma, F)$ holds, respectively.

The inference rules of Fig. 2.1 give one case per production. Note how the context is used in (and only in) the case $\Gamma \vdash_\Sigma x : \texttt{term}$.

Example 6.33 (Static Analysis). The static analysis from Bonus Exercise 1 uses context-sensitive induction. The context stores the declared identifiers and for every identifier its type.

The return type of the functions varies: For example, $f_{\texttt{program}}(\Gamma, E)$ is a boolean; and $f_{\texttt{expression}}(\Gamma, E)$ is a function that takes the expected type of E and returns a boolean.

Example 6.34 (Interpretation). The interpretation from Bonus Exercise 1 uses context-sensitive induction. The context stores the declared identifiers and for every identifier its type and its current value.

The return type of the functions varies: $f(\texttt{program})$ is irrelevant/empty and $f(\texttt{expression})$ is the set containing all string, integer, and boolean values of the underlying programming language.

6.5 Context-Free Induction in First-Order Logic

The term language of first-order logic is great for working with context-free grammars. In fact, we can think of FOLEQ as an abstract syntax framework for context-free grammars.

There is only one caveat: There may only be one non-terminal symbol. However, there is a straightforward generalization of FOLEQ (called typed FOLEQ or sorted FOLEQ) that can use a separate type for each non-terminal symbol.

But the restriction to a single non-terminal is not so bad: It already covers some of the most important context-free grammars.

Example 6.35 (Natural Numbers (Continuing Ex. 6.29)). The grammar

$$N \quad ::= \quad 0 \quad | \quad succ(N)$$

corresponds the FOLEQ signature from Ex. 1.33.

Example 6.36 (Words). The language of words over an alphabet $A = \{a_1, \ldots, a_n\}$ can be written as the context-free grammar

$$W \quad ::= \quad \varepsilon \quad | \quad a_1 W \quad | \quad \ldots \quad | \quad a_n W$$

It corresponds to the FOLEQ signature using
- nullary function symbol ε
- unary function symbols $\boldsymbol{a_1}$, $\ldots$, $\boldsymbol{a_n}$, written as prefix without brackets, i.e., $\boldsymbol{a}w$ instead of $\boldsymbol{a}(w)$.

6.5.1 Simple Definitions in First-Order Logic

FOLEQ does not permit inductive definitions directly. In fact, FOLEQ signatures do not permit *any* definition at all. But we can give almost any definition indirectly using axioms in FOLEQ theories.

Constant Definitions For some term $\vdash_\Sigma t : \texttt{term}$, the definition $c := t$ can be written in a FOLEQ theory by adding
- a nullary function symbol c,
- the axiom $c \doteq t$

Example 6.37 (Continuing Ex. 6.35). We define $1 := \mathit{succ}(0)$ by adding
- the nullary function symbol 1,
- the axiom $1 \doteq \mathit{succ}(0)$

Function Definitions For some term $x_1, \ldots, x_n \vdash_\Sigma t : \texttt{term}$, the definition f is the function that maps $x_1, \ldots, x_n$ to t can be written in a FOLEQ theory by adding
- an n-ary function symbol f,
- the axiom $\forall x_1 \ldots \forall x_n \; f(x_1, \ldots, x_n) \doteq t$

Note that constant definitions can be seen of the special case of a function definition where $n = 0$.

Example 6.38 (Continuing Ex. 6.36). We define the doubling ww of a word using the term $w \vdash_\Sigma ww : \texttt{term}$ by adding
- the unary function symbol *double*,
- the axiom $\forall w \; \mathit{double}(w) \doteq ww$

Predicate Definitions For some formula $x_1, \ldots, x_n \vdash_\Sigma F : \texttt{form}$, the definition p is the property that holds for $x_1, \ldots, x_n$ if F holds can be written in a FOLEQ theory by adding
- an n-ary predicate symbol p,
- the axiom $\forall x_1 \ldots \forall x_n \; p(x_1, \ldots, x_n) \leftrightarrow t$

Example 6.39 (Continuing Ex. 6.35). We define the property *nonzero* of a natural number using the formula $n \vdash_\Sigma \neg n \doteq 0 : \texttt{form}$ by adding
- the unary predicate symbol *nonzero*,
- the axiom $\forall n \; \mathit{nonzero}(n) \leftrightarrow \neg n \doteq 0$

6.5.2 Inductive Definitions in First-Order Logic

We obtain much more flexible options for definitions if we use induction. The basic idea is the same: We use axioms to capture the content of the definition. More precisely, we use one axiom for each case of the induction.

Inductive Function Definitions Functions are defined by adding a function symbol and then one axiom for every case.

Example 6.40 (Continuing Ex. 1.34, 6.3 and 6.35). We define the addition $m + n$ of natural numbers by adding
- the binary function symbol $+$ (written infix),
- the axioms $\forall n \; 0 + n \doteq n$ and $\forall m \forall n \; \mathit{succ}(m) + n \doteq \mathit{succ}(m + n)$.

And we define the multiplication $m \cdot n$ of natural numbers by adding

- the binary function symbol $\cdot$ (written infix),
- the axioms $\forall n\ 0 \cdot n \doteq 0$ and $\forall m \forall n\ succ(m) \cdot n \doteq (m \cdot n) + n$.

Example 6.41 (Continuing Ex. 6.15 and 6.36). We define the reversal of a word by adding
- the unary function symbol R (written as postfix superscript, i.e., w^R),
- the axioms $\forall w\ \boldsymbol{\varepsilon}^R \doteq \boldsymbol{\varepsilon}$ and $\forall w\ (\boldsymbol{a}w)^R \doteq w^R\boldsymbol{a}$ for all unary symbols $\boldsymbol{a}$ from Ex. 6.36.

Inductive Predicate Definitions Properties/predicates are defined by adding a predicate symbols and one axiom for every case.

Example 6.42 (Continuing 6.35). We define the less-or-equal predicate $\leq$ between natural numbers by adding
- the binary predicate symbol $\leq$ (written infix),
- the axioms

$$\forall n\ 0 \leq n \leftrightarrow \mathit{true} \qquad \text{and} \qquad \forall m \forall n\ succ(m) \leq n \leftrightarrow \exists x\ n \doteq succ(x) \wedge m \leq x$$

Remark 6.43. Of course, we can simply write $0 \leq n$ instead of $0 \leq n \leftrightarrow \mathit{true}$. We use the longer version here to emphasize the general shape of inductive predicate definitions.

Mutually Inductive Definitions We can also state mutually inductive definitions of functions or predicates. We simply add multiple function/predicate symbols at once and give one axiom per case.

Example 6.44 (Continuing 6.35). We define the predicates *even* and *odd* on natural numbers by adding
- the unary predicate symbols *even* and *odd*,
- the axioms

$$\mathit{even}(0) \leftrightarrow \mathit{true} \qquad \text{and} \qquad \forall n\ \mathit{even}(succ(n)) \leftrightarrow \mathit{odd}(n)$$
$$\mathit{odd}(0) \leftrightarrow \mathit{false} \qquad \text{and} \qquad \forall n\ \mathit{odd}(succ(n)) \leftrightarrow \mathit{even}(n)$$

6.5.3 Inductive Proofs in First-Order Logic

We can now finally understand the motivation behind the Peano axioms from Ex. 1.33:

Example 6.45 (Induction Schemata in First-Order Logic). The Peano axioms from Ex. 1.33 arise as follows:

- The function symbols 0 and *succ* formalizes the cases 1 and 2 of Def. 6.1.
- The axioms for injectivity of *succ* and starting point 0 formalize the cases 3a of Def. 6.1. Respectively, they capture that no two successors are equal and that no successor is equal to 0.
- The axiom schema of induction approximately formalizes case 3b of Def. 6.1. It expresses the intuition that 0 and all successors make up the whole natural numbers.

Thus, the Peano axioms systematically formalize Def. 6.1 in FOLEQ.

Along these lines, we can define induction schemata in FOLEQ for all context-free grammars with a single non-terminal. Then we can revisit the examples of inductive proofs from Sect. 6.1, 6.2, and 6.3 and formally carry them out in FOLEQ.

Remark 6.46 (Incompleteness of Induction). Note that Ex. 6.45 says "approximately formalizes". The problem is that the axiom schema of induction yields only countably many axioms (because there are only countably many formulas). But there are uncountably many things we can do with the natural numbers (e.g., uncountably many different functions out of $\mathbb{N}$). Therefore, FOLEQ cannot fully capture case 3b of Def. 6.1.

Indeed, as we know from Ex. 3.44, even though the theory of Ex. 6.45 is complete (i.e., can prove or disprove every formula), there are non-standard models that satisfy the exact same FOLEQ formulas as the intended standard model SN.

Moreover, and even worse, we know that Peano arithmetic – i.e., the union of the theories from Ex. 6.40 and 6.45 – is not even a complete theory: There are formulas that are true in the standard model but can be neither proved nor disproved in FOLEQ. The proofs of these formulas require induction hypotheses that FOLEQ's Peano arithmetic cannot express.

Chapter 7

Summary

Logics Intuitively, logics are formal language with a mathematical semantics. This is different from natural language (whose semantics is often unclear) and other formal languages used in computer science such as programming or data description languages (whose semantics is usually given by a textual specification).

Logics consist of syntax, proof theory, and model theory as well as soundness and (ideally) completeness theorems. The syntax defines the formal language. The proof and model theory define one semantics each. The soundness and completeness theorems show that the proof and the model theoretical semantics are equivalent.

Syntax The syntax consists of a set of signatures and one formal language for each signature.
Each signature fixes some globally available symbols, whose meaning is open to interpretation.
The formal language is typically context-sensitive and defined by a context-free grammar and a context-sensitive inference system for well-formedness. The terminal symbols of the grammar consist of the symbols of the signature (e.g., function and predicate symbols) and the logical symbols (e.g., $\wedge\vee\to\neg\forall\exists\doteq$).

The formal language includes in particular the set of sentences $\mathbf{Sen}(\Sigma)$ for every signature Σ.
Theories consists of a signature Σ and a set $\Theta \subseteq \mathbf{Sen}(\Sigma)$ of axioms.
The semantics of the logic is given by the consequence relation between theories Θ and sentences $F \in \mathbf{Sen}(\Sigma)$. If the relation holds, F is called a theorem.

Model Theory The model theory defines models $I \in \mathbf{Mod}(\Sigma)$ for every signature.
For each model, the function $[\![-]\!]^I$ is defined by context-sensitive induction on the syntax of Σ. It maps every word to its interpretation in I.
Sentences F are interpreted as truth values $[\![F]\!]^I \in \{0,1\}$. If $[\![F]\!]^I = 1$, we say that F is true in I, satisfied by I, or holds in I.
The model theoretical consequence relation is written $\models \Sigma\Theta F$. It holds if all models that satisfy all axioms also satisfy F.

The purpose of Σ-models is that they define different interpretations for the symbols of Σ. That way the same syntax can be used to talk about lots of different phenomena represented by different models.
Moreover, the syntax is available to computer implementation, whereas the most models are not. Thus, the symbols of Σ act as abstract representatives for objects that cannot be implemented themselves, and each model bundles a set of concrete objects (one for each symbol of Σ).

Proof Theory The proof theory defines proofs for every signature.
Proofs are trees whose nodes are labelled with judgments and which are formed by applying inference rules.
The proof theoretical consequence relation is given by the judgment $\Theta \vdash_\Sigma F$. The relation holds if the judgment has a proof.

The purpose of proofs is to capture the intuition of reasoning and consequence by a finite set of rules. This permits the systematic and mechanical exploration of the consequence relation.
In particular and contrary to the model theory, it can be implemented.

Soundness and Completeness A logic is sound if proof theoretical consequence implies model theoretical consequence.
Soundness is useful because it legitimizes the mechanic reasoning performed by the proof rules.

A logic is complete if model theoretical consequence implies proof theoretical consequence.
Alternatively, we can say that there must be enough proofs to prove all the model theoretical theorems, or that there must be enough models to provide counter-examples for all the proof theoretical non-theorems.

Propositional Logic Propositional logic is the simplest useful logic.
Its signatures declare only names of unknown formulas, and its logical symbols are only $\wedge\vee\rightarrow\neg$.

Models interpret the signature symbols as truth values.
Proofs can be formed in various ways, in particular using the natural deduction calculus.

Propositional logic PL is sound and complete.
PL is special in that its formal language is context-free and its consequence relation is decidable.

First-Order Logic First-order logic is a generalization of propositional logic. It adds the logical symbols $\forall\exists$ and depending on the definition also $\doteq$.
The grammar for formulas, the context-sensitive interpretation function, and the natural deduction proof rules are extended appropriately for the new symbols.

The main addition of FOL is that, in addition to truth values, it talks about data.

Syntactically, the data is described by terms. These are formed from function symbols with certain arities and variables. The former are introduced by the signature, the latter by the quantifiers $\forall$ and $\exists$.

Model theoretically, the data is described by the universe: a non-empty set $\texttt{term}^I$ provided by the model. Models interpret n-ary function symbols as n-ary functions on the universe.

Additionally, signatures may introduce predicate symbols, which describe properties of the data. Predicate symbols provide additional formulas if applied to terms.
Models interpret n-ary predicate symbols as n-ary relations on the universe.

Proof theoretically, there is no special treatment necessary for data.

First-order logic FOL (or FOLEQ if $\doteq$ is used) are sound and complete. Moreover, FOLEQ is essentially the simplest possible logic that is still sound and complete.

FOLEQ can be used to define a wide variety of data types by giving an appropriate theory. Examples include the theories of monoids, groups, rings, fields, relations, orders, and lattices. The soundness and completeness properties guarantee that e.g., everything we prove in the theory of fields holds for every individual field.

Incompleteness While logics such as FOLEQ are great for representing abstract data types (e.g., fields), it has two major limitations when it comes to representing concrete data types (e.g., the natural numbers).

Firstly, we cannot give a theory whose models are exactly the natural numbers. There are non-standard models that satisfy exactly the same FOLEQ-formulas as the natural numbers without being isomorphic to them.

Secondly, the set of FOLEQ-formulas that hold in the natural numbers is not recursively enumerable. Therefore, no sound and complete logic can express them with a recursively enumerable set of axioms.

According results hold for all more complex concrete data types.

Formal Languages Involved in Logic The formal languages of logic are usually context-sensitive, and the sublanguages containing only the theorems are usually only recursively enumerable but not decidable.

Formal Language	Classification
formulas over a fixed	
PL-signature	context-free
FOL or FOLEQ-signature in any context	context-free
FOL or FOLEQ-signature in a fixed context	context-sensitive
theorems over a fixed consistent[1]	
PL-theory	decidable[2] (but not context-sensitive)
FOL or FOLEQ-theory	recursively enumerable[3] (but usually not decidable)

Induction Induction permits defining functions out of a formal language. The most important induction principle is the one for context-free languages, where we give one case for every production.

The context-free induction principle can be generalized to the context-sensitive induction principle. Here we take a context as an additional argument.

The functions for substitution application and the for interpretation in a model are defined by context-sensitive induction on the formulas of FOLEQ.

[1]If the theory is inconsistent, the set of theorems is trivially regular.
[2]Technically, the set of axioms must be finite.
[3]Technically, the of axioms must be recursively enumerable. If the theory is also complete, the set of theorems is decidable.

Bibliography

[And86] P. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Academic Press, 1986.

[Can83] G. Cantor. Grundlagen einer allgemeinen Mannigfaltigkeitslehre. Ein mathematisch-philosophischer Versuch in der Lehre des Unendlichen. *Mathematische Annalen*, 1883.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[Fra22] A. Fraenkel. Zu den Grundlagen der Cantor-Zermeloschen Mengenlehre. *Mathematische Annalen*, 86:230–237, 1922. English title: On the Foundation of Cantor-Zermelo Set Theory.

[Fre79] G. Frege. *Begriffsschrift: eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. 1879.

[Gal86] J. Gallier. *Logic for Computer Science*. hr, 1986.

[GB92] J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

[Gen34] G. Gentzen. Untersuchungen ber das logische Schließen. *Math. Z.*, 39, 1934. English title: Investigations into Logical Deduction.

[Göd30] K. Gödel. Die Vollständigkeit der Axiome des Logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37:349–360, 1930. English title: The Completeness of the Axioms of the Logical Calculus of Functions.

[Göd31] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931. English title: On Formally Undecidable Propositions Of Principia Mathematica And Related Systems.

[Hil26] D. Hilbert. Über das Unendliche. *Mathematische Annalen*, 95:161–90, 1926.

[Pea89] G. Peano. The principles of arithmetic, presented by a new method. 1889.

[Rob50] A. Robinson. On the application of symbolic logic to algebra. In *Proceedings of the International Congress of Mathematicians*, pages 686–694. American Mathematical Society, 1950.

[Smu95] R. Smullyan. *First-Order Logic*. Dover, second corrected edition, 1995.

[TV56] A. Tarski and R. Vaught. Arithmetical extensions of relational systems. *Compositio Mathematica*, 13:81–102, 1956.

[WR13] A. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, 1913.

[Zer08] E. Zermelo. Untersuchungen ber die Grundlagen der Mengenlehre I. *Mathematische Annalen*, 65:261–281, 1908. English title: Investigations in the foundations of set theory I.