Many-Sorted Languages

A. Bernhard Zeidler

思考

Contents

This material may be referenced, reproduced and distributed in whole or in part, in any medium physical or electronic, provided that (1) the terms of the *Open Publication License* are adhered to, that (2) this license or an incorporation of it by reference is displayed in the reproduction and that (3) the original author and a reference to the site of this document is clearly stated. I.e. any reproduction or distribution has to meet all the conditions set forth in the *Open Publication License*, v1.0 or later, given in section 9 (the latest version is presently available at www.opencontent.org/openpub). Distribution or derivation of the work for commercial purposes is prohibited, unless prior permission is obtained from the copyright holder.

1 What do we have here?

Overview

In this text we introduce many-sorted languages as a more natural way to formalize mathematics than the common one-sorted logic. While this is not really necessary from a rigid point of view (all mathematics can be covered with set-theory, which can be done in the one-sorted language of sets) there are many good reasons to use many-sorted languages instead:

- Almost all mathematics speaks of several sorts of objects: Points and lines in plane geometry, vectors and scalars in linear algebra and even sets and classes in formal set theory. It stands to reason to formalize this as it is.
- Many-sorted languages are no more complicated than one-sorted languages, and it only takes little longer to formulate this machinery.
- While it is possible to boil down many-sorted languages with finitely many sorts into a one-sorted language and use predicates (i.e. a 1-ary relation symbol saying *I am of sort "i"*) it is harder to prove that this construction preserves all the theorems (like Gödel's Completeness Theorem) than to start from scratch.

So what we try to do here is to present an introduction into many-sorted languages and models and to formulate and prove the most important theorems of formal logic, like the correctness, completeness and compactness theorems. We will also present examples of common many-sorted languages along the way, including the language of modules over commutative rings, of ring-extensions and that of sets and classes. If you are not familiar with naive mathematical logic yet, read remark 33 first.

My approach to formal languages follows the extremely elegant and efficient path my esteemed teacher Prof. Ulrich Felgner of the university of Tübingen laid out for one-sorted languages. Part of my thesis required a generalization to many-sorted languages. As he never wrote a textbook on this himself, I try to preserve part of his marvelous insights herein.

Introduction

David Hilbert gave a set of axioms for Euclidean Geometry, i.e. the geometry of points, lines and planes in 3-dimensional space. His first axiom reads as: "For any two points there exists a straight line passing through them". In more formal terms this would read as

$$\forall\, P \text{ point } \forall\, Q \text{ point } \exists\, g \text{ line } : (P \in g \text{ and } Q \in g)$$

So let us take a closer look at this: We have 3 sorts of objects: points, lines and planes. Lines and planes consist of points, hence there is a relation, e.g. $P \in g$ that designates whether a point P is contained in the line g. Note that this relation is between points and lines. There is another such relation between points and planes. They are usually denoted by the same symbol $\in$ but from a formal point of view these are two distinct relations.

This becomes even more important, if we regard the intersection $\cap$ of lines. It could be formalized as a function that turns two lines into a point (with $\emptyset$ being a special point that we could pick up as a constant). Likewise the intersection of planes yields a line (with another $\emptyset$ being a special line). We even have an intersection of a line and a plane, which should be a point. That is we need to distinguish between three separate functions:

$$\begin{array}{rcccc} \cap_L & : & \{\,\text{lines}\,\} \times \{\,\text{lines}\,\} & \to & \{\,\text{points}\,\} \\ \cap_P & : & \{\,\text{planes}\,\} \times \{\,\text{planes}\,\} & \to & \{\,\text{lines}\,\} \\ \cap_T & : & \{\,\text{lines}\,\} \times \{\,\text{planes}\,\} & \to & \{\,\text{points}\,\} \end{array}$$

What we find here is that functions start from (one or more) objects of predefined sort(s) and yield another object of another, fixed sort. Like relations you cannot enter just any object of any sort into a function.

As always the quantifiers $\forall$ and $\exists$ translate into *for all* and *there is* but we note, that they always apply to one sort of objects only, e.g. for all points P there is this and that. Hence every sort has its own quantifiers, too.

All these considerations are in no way artificial - this is what mathematicians do all the time: Thinking about different types of objects and their interaction. But before we venture on to giving a formal definition of the structure of many-sorted languages let us present another example from a different field: Linear Algebra. In the theory of vector-spaces (or modules over a ring) we have a base field F and a vector-space V. The base field features the operations of addition and multiplication. The vector-space only has an addition, but there also is a scalar multiplication between scalars and vectors. Altogether we have 4 binary functions:

$$\begin{array}{rcccc} +_f & : & F \times F & \to & V \\ \cdot_f & : & F \times F & \to & F \\ +_v & : & V \times V & \to & V \\ \cdot_s & : & F \times V & \to & V \end{array}$$

Note that we also have constants, the neutral elements 0_f and 1_f in F and 0_v in V. One of the properties required for a vector-space is, that for any two scalars $a, b \in F$ and for any vector $x \in V$ we have $(a+b)x = (ax)+(bx)$.

Note that in this equation the first addition is $+_f$ and the second is $+_v$. But this only is the usual operator-overloading commonplace in mathematics. What we want to stress here, is the use of quantifiers: What has been covered in $a \in F$ or $x \in V$ will turn into different quantifiers in a many-sorted language. So this formula would read, as:

$$\forall_f a \;\; \forall_f b \;\; \forall_v x \;:\; (a +_f b) \cdot_s x \;=\; (a \cdot_s x) +_v (b \cdot_s x)$$

It might be a little cumbersome to track precisely which + belongs to which sort, if you are familiar with these expressions there is no loss in omitting the indices. But this is not the case, when it comes to quantifiers! It is safe to rewrite $\forall_f a$ as $\forall a \in F$ but it is important to note, that it only applies to all objects of the sort F. So in the end you will read the following formula (next page) in a textbook about linear algebra, but remember - this is natively a formula of in two-sorted language!

$$\forall\, a \in F \;\; \forall\, b \in F \;\; \forall\, x \in V \;:\; (a+b)x \;=\; (ax) + (bx)$$

From these two examples it should be apparent, that the natural way to formalize mathematics is a structure that allows us to speak of different sorts of objects. every sort has its own quantifiers and constants. Relations and functions can link objects of different sorts but are sort-dependent, as well: You cannot insert a scalar into a slot where a vector is supposed to be.

Back to Where it all Began

If you, dear reader, are still in doubt what all that fuzz is about, why we need to further formalize mathematics anyway, let us relay Russel's Paradox that crashed Cantor's notion of sets: Let us take a look at the set of all set, that do not contain itself, formally $\Omega := \{\, X \mid X \notin X \,\}$. As of Cantor's notion this is a totally legit construction: It is a set of objects X that satisfy a reasonable condition $X \notin X$.

So the question is: Is $\Omega \in \Omega$? If "yes" then we would have $\Omega \notin \Omega$, by the definition of Ω, a contradiction. If "no" this is $\Omega \notin \Omega$ and hence we should have had $\Omega \in \Omega$, another contradiction. Either way the coin lands, we arrive at a contradiction - so what is wrong?

Alas, the answer is not a short one: Cantor's notion is just too far reaching to be free of such contradictions. What we need to do is what has been called *reduction of size*. That is we have to start with very simple sets, $\emptyset$ for example, and then slowly work ourselves up to more complex sets. Given two (or more) sets the constructions allowed are intersections, unions and power sets. These then turn up other nice sets. We can even build up Cartesian products of sets this way. So whatever we can build up using these constructions is safe. So mathematicians build up the naturals, integers, rationals, reals, complex numbers and so on from scratch - only using $\emptyset$ and these constructions. All this is done in the formal language of sets. This is a one-sorted language, all the constants and variables are considered sets.

However it is tempting to talk about the *set of all sets* - this essentially is what Ω has been in the first place. Or at least the set of all groups. But it can be shown that any (non-empty) set can be equipped with a group structure, so this does not help either.

But category theory talks about such concepts all the time - they even have functors, that map an object from one category to another (e.g. we start with a commutative ring R and end up with the topological space $\mathrm{Spec}(R)$). So how do they avoid these paradoxa? Simple: By avoiding the word *set* at the wrong times.

It is perfectly fine to speak about the *class* of all sets. The problem was, that Cantor allowed a language that could speak of itself. If Ω is a class (not of the sort *set*) then the relation $\Omega \in \Omega$ is forbidden. So in modern set and category theory there are two sorts: sets and classes. Some classes are small enough to be sets themselves, but not necessarily so.

There are two ways to tackle this from a formal point of view: First of all we can take the variables to be classes and introduce a predicate for classes saying *I also am a set* or we could use a two-sorted language, with one sort being sets, the other being classes. Both approaches have their charms.

2 Defining Many-Sorted Languages

Let us start by giving a definition of formal languages with several sorts. Admittedly the definition is somewhat lengthy but nevertheless understandable at first glance. The simplicity of the definition is due to a peculiarity that it has in common with for example Cantor's "definition" of sets: The definition is (more or less) informal and appeals to the readers common sense, not some advanced mathematical knowledge.

Of course there is some danger to this - as we have pointed out Cantor's definition allowed to construct Russel's Paradox. In our case, however, things are free of hazards, as we only give rules that can be formalized easily on the basis of naive set theory. Yet we wish to omit a precise formalization, as this wouldn't make anything clearer, but only obfuscate the simplicity of the formalism.

The first object we require are the words over a given alphabet. Formal languages will then be defined by fixing a certain alphabet and giving a collection of rules that determine whether any given word belongs to the language or not. Intuitively speaking let $\mathcal{A}$ be any collection of symbols satisfying the following two properties:

- any two different symbols of $\mathcal{A}$ look different and
- no symbol of $\mathcal{A}$ is part of another symbol of $\mathcal{A}$

Then such a collection is said to be an **alphabet**. If we write some (i.e. finitely many) symbols of $\mathcal{A}$ in a row then this sequel of symbols is said to be a **word** over $\mathcal{A}$. The collection of all words over $\mathcal{A}$ is denoted by $\mathcal{A}^*$. More formally

$$a_1 a_2 \dots a_k \in \mathcal{A}^* \quad :\Longleftrightarrow \quad a_1 \in \mathcal{A}, \dots, a_k \in \mathcal{A}$$

Nota that the above two properties guarantee that any word can be read unambiguously. If for example we had two symbols $a = \cdot$ and $b = \cdot\cdot$ then the word $\cdot\cdot\cdot$ could be read as ab, ba or even aaa. Yet this may not occur due to the second property. The idea of words over a given alphabet should be clear enough now, but here's a rigid definition based on naive set theory:

Definition 1:
Let $\mathcal{A}$ be any non-empty set, that we use as a collection of symbols. Then the set $\mathcal{A}^*$ of words over $\mathcal{A}$ is be defined to be

$$\mathcal{A}^* \quad := \quad \bigcup_{k \in \mathbb{N}} \mathcal{A}^k$$

That is a word $a = a_1 a_2 \dots a_k$ is just an n-tuple $a = (a_1, a_2, \dots, a_k) \in \mathcal{A}^k$ for some non-negative integer $k \in \mathbb{N}$. The case $k = 0$ includes the **empty word** which is denoted by ε. The number k with $a \in \mathcal{A}^k$ is called the **length** $\ell(a)$ of the word, denoted by

$$\ell(a_1 \dots a_k) \quad := \quad k$$

The **concatenation** ab of words $a = a_1 a_2 \dots a_m$ and $b = b_1 b_2 \dots b_n \in \mathcal{A}^*$ is just the word $ab = a_1 a_2 \dots a_m b_1 b_2 \dots b_n \in \mathcal{A}^*$. Doing this $\mathcal{A}^*$ becomes a monoid under the concatenation, $\ell : \mathcal{A}^* \to \mathbb{N}$ is a homomorphism of such.

If $b \in \mathcal{A}^*$ is **part of** $d \in \mathcal{A}^*$ we write $b \mid d$, formally that is

$$b \mid d \quad :\Longleftrightarrow \quad \exists\, a, c \in \mathcal{A}^* \,:\, abc = d$$

And if we wish to refer to the i-th symbol of the word a (where $i \in 1 \ldots \ell(a)$ is any number from 1 to the length of a) we write $a[i]$, formally

$$a[i] \,:=\, a_i \,\in\, \mathcal{A} \quad \text{for} \quad a \,=\, a_1 \ldots a_k \,\in\, \mathcal{A}^*$$

Example 2:
The Latin alphabet consists of $2 \cdot 26$ symbols, namely the common letters

$$\mathcal{A} \,:=\, \{\, a, b, c, \ldots, z, A, B, C, \ldots, Z \,\}$$

Thus *word* $\in \mathcal{A}^*$ is a word over the Latin alphabet and the first sentence of the fourth gospel of John

In the beginning, before the creation of the world, he was, who is the word.

is a sequel of 15 words separated by interpunctional signs. Note however that these signs (space, comma and full stop) do not even belong to the alphabet. If the Latin alphabet was appended by these three symbols, then the whole sentence would be a single word over this appended alphabet.

Definition 3:
A many-sorted language $\mathcal{L}$ is based on four ingredients: a collection $\mathcal{C}$ of constant symbols, a collection $\mathcal{F}$ of function symbols and a collection $\mathcal{R}$ of relation symbols. Hereby we require that all these symbols can be distinguished from one and another. Further any symbol has a certain signature of sorts as described below. The collection of all sorts available is thereby denoted by I. Then we can define what we mean by a (many-sorted, first order) formal language $\mathcal{L} := \mathcal{L}_I(\mathcal{C}, \mathcal{F}, \mathcal{R}, \text{sort})$ in several steps:

- **the symbols**
  The alphabet $\mathcal{A}$ of the language $\mathcal{L}$ consists of the following symbols
  - a **left bracket** (and a **right bracket**)
  - two logical symbols: the **negation** $\neg$ and **implication** $\to$
  - for every sort $i \in I$ a symbol of **equality** $=_i$, a **universal quantifier** $\forall_i$ and an **existential quantifier** $\exists_i$
  - for each **sort** $i \in I$ and every natural number $j \in \mathbb{N}$ we fix a **variable symbol** $x_{i,j}$ and this is assigned $\text{sort}(x_{i,j}) := i \in I$.
  - every **constant symbol** $c \in \mathcal{C}$ given, where c is supposed to already have been assigned a certain $\text{sort}(c) \in I$.
  - every **function symbol** $f \in \mathcal{F}$ given, where f is supposed to already have been assigned a signature of the form $\text{sort}(f) \in I^{k+1}$ for some $1 \leq k \in \mathbb{N}$.
  - every **relation symbol** $R \in \mathcal{R}$ given, where R is supposed to already have been assigned a signature of the form $\text{sort}(R) \in I^k$ for some $1 \leq k \in \mathbb{N}$.

We will now define what we mean by a *term* or *formula* of the language $\mathcal{L}$. I.e. we decree that some words over the above alphabet will be called terms, others will be called formulae. However we will not give an explicit definition, but instead give rules how such words may be obtained. Then a term (or formula respectively) is simply a word which was created using the rules given. Further any term will be assigned a sort in I simultaneously to its construction.

- **the terms**
  We call a word $t \in \mathcal{A}^*$ a **term** of the language $\mathcal{L}$ iff it can be generated by applying the following rules finitely many times:

  (T1) any variable symbol x of $\mathcal{L}$ already is a term of $\mathcal{L}$, and this term is assigned the sort $\mathrm{sort}(x)$.

  (T2) any constant symbol c of $\mathcal{L}$ already is a term of $\mathcal{L}$, and this term is assigned the sort $\mathrm{sort}(c)$.

  (T3) for any function symbol f of $\mathcal{L}$ with $\mathrm{sort}(f) = (i_1, \ldots, i_k, i_{k+1})$ and for any (previously generated) terms $t_1, \ldots, t_k \in \mathrm{term}(\mathcal{L})$ of $\mathcal{L}$, having the sorts $\mathrm{sort}(t_1) = i_1, \ldots, \mathrm{sort}(t_k) = i_k$ we decree that the word $ft_1 \ldots t_k \in \mathcal{A}^*$ is a term of $\mathcal{L}$ of $\mathrm{sort}(ft_1 \ldots t_k) := i_{k+1}$.

- **atomic formulae**
  We call a word $\varphi \in \mathcal{A}^*$ an **atomic formula** of the language $\mathcal{L}$ iff it is of one of the following two types

  (A1) for any two terms s, t of the same sort $i := \mathrm{sort}(s) = \mathrm{sort}(t)$, the word $s =_i t \in \mathcal{A}^*$ is an atomic formula of $\mathcal{L}$.

  (A2) for any relation symbol R of the sort $\mathrm{sort}(R) = (i_1, \ldots, i_k)$ and any terms $t_1, \ldots, t_k \in \mathrm{term}(\mathcal{L})$ of the sorts $\mathrm{sort}(t_1) = i_1$ up to $\mathrm{sort}(t_k) = i_k$ the word $Rt_1 \ldots t_k \in \mathcal{A}^*$ is an atomic formula of $\mathcal{L}$.

- **formulae**
  We call a word $\varphi \in \mathcal{A}^*$ a **formula** of the language $\mathcal{L}$ iff it can be generated by applying the following rules finitely many times

  (F1) any atomic formula φ of $\mathcal{L}$ already is a formula of $\mathcal{L}$.

  (F2) for any two previously generated formulae φ and ψ of $\mathcal{L}$, the words $(\neg\varphi) \in \mathcal{A}^*$ and $(\varphi \to \psi) \in \mathcal{A}^*$ are formulae of $\mathcal{L}$, too.

  (F3) for any variable symbol x of $\mathcal{L}$ of the sort $i := \mathrm{sort}(x)$ and any previously generated formula φ of $\mathcal{L}$, the words $(\exists_i x\, \varphi) \in \mathcal{A}^*$ and $(\forall_i x\, \varphi) \in \mathcal{A}^*$ are formulae of $\mathcal{L}$, too.

Let us denote the set $\mathrm{var}(\mathcal{L}) := \{ x_{i,j} \mid i \in I, j \in \mathbb{N} \}$ of variable symbols and the set $\mathrm{quant}(\mathcal{L}) := \{ \exists_i \mid i \in I \} \cup \{ \forall_i \mid i \in I \}$ of quantifier symbols of $\mathcal{L}$. The set of all terms of $\mathcal{L}$ will be denoted by $\mathrm{term}(\mathcal{L})$, the set of atomic formulae of $\mathcal{L}$ by $\mathrm{atom}(\mathcal{L})$ and - not surprisingly - the set of all formulae of $\mathcal{L}$ by $\mathrm{form}(\mathcal{L})$. By rule (F1) we have $\mathrm{atom}(\mathcal{L}) \subseteq \mathrm{form}(\mathcal{L})$. Now the set $\mathcal{L}$ itself is finally defined by

$$\mathcal{L} \;:=\; \mathrm{term}(\mathcal{L}) \cup \mathrm{form}(\mathcal{L}) \;\subseteq\; \mathcal{A}^*$$

This definition only provides the minimum basis of the mathematical formalism, of course. It is presented in this way to ensure unique readability, but is not yet perfectly suited to be truly used in mathematics. Thus in formal logic the formulae regarded have a tendency to become very long and intricate. In order to remain on top we need some to put them into a form easier to read and hence introduce the following notational conventions:

Notation 4:
Let $\mathcal{L} = \mathcal{L}_I(\mathcal{C}, \mathcal{F}, \mathcal{R}, \text{sort})$ be a many-sorted language, then we introduce the following notational conventions concerning terms and formulae

(i) If I consists of one element only $I = \{i\}$, we say that $\mathcal{L}$ is a one-sorted language. In this case we don't need the sort function and can omit all the indexes i on variable symbols $x_{i,j}$, equality relations $=_i$ and the quantifiers $\forall_i$ and $\exists_i$.

(ii) By a theorem of Post any junctor can be expressed using a combination of $\neg$ and $\rightarrow$. Therefore we can use any junctor as an expression of $\mathcal{L}$, defining its usage by a logically equivalent formula composed of $\neg$ and $\rightarrow$. In this text we will frequently employ the abbreviations

$$\begin{array}{lll} \varphi \vee \psi & :\Longleftrightarrow & (\neg\varphi) \rightarrow \psi \\ \varphi \wedge \psi & :\Longleftrightarrow & \neg(\varphi \rightarrow (\neg\psi)) \\ \varphi \leftrightarrow \psi & :\Longleftrightarrow & (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \end{array}$$

NOTA we distinguish between single-stroke arrows, as in $\varphi \rightarrow \psi$ that are part of the formal language and double-stroke arrows, as in (x is a cat) $\Longrightarrow$ (x is a mammal), which belong to the meta-language. As in the example of the implication here, we will build our formalism of deduction in such a way, that $\rightarrow$ will behave just like $\Longrightarrow$.

(iii) We follow the usual conventions concerning the bracketing of terms and formulae, i.e. in order to save some brackets, we decree that $\neg$ is of higher priority than $\wedge$ and $\vee$, which in turn are considered of higher priority than $\rightarrow$ and $\leftrightarrow$. As an example the formula

$$\neg\varphi \vee \psi \ \rightarrow \ \chi \wedge \omega$$

is a shorter notation for the (more obfuscated but correct) formula

$$(((\neg\varphi) \vee \psi) \ \rightarrow \ (\chi \wedge \omega))$$

(iv) We apply brackets to functions, i.e. we write $f(t_1, \ldots, t_k)$ instead of $ft_1 \ldots t_k$. The same is true for relations - we write $R(t_1, \ldots, t_k)$ instead of $Rt_1 \ldots t_k$. An exception to this are binary relations which are usually written in the form sRt instead of Rst. (NOTA that the comma here doesn't even belong to the symbols of the formal language).

Regard the (nonsensical) formula $\varphi := \forall y\,(x = y \vee \exists x\,(x = z))$ then φ contains the variable symbols x, y and z. Yet there is a difference: The variable symbol y is quantified over and hence cannot be "seen from the outside".

Hence y is called a *bounded* variable of φ, whereas x and z are said to be *free* variables of φ. We will emphasize this by writing $\varphi(x, z)$ when needed.

But this is not the sole difference, in the sub-formula $\exists\, x\,(x = z)$ the variable symbol x is bounded and hence only the first occurance of x in φ is free, the second is bounded. Hence if we want to *substitute* the variable x by a term t this is only allowed in the first instance. Hence the substitution of x by t in φ turns out to be $\varphi[x : t] = \forall\, y\,(t = y \vee \exists\, x\,(x = z))$. Let us take a general look at these notions in the following:

Definition 5:
Let again $\mathcal{L} = \mathcal{L}_I(\mathcal{C}, \mathcal{F}, \mathcal{R}, \text{sort})$ be a many sorted language, then we introduce the following notions for terms and formulae of $\mathcal{L}$. As these were defined recursively it stands to reason that we may revert this recursion and decompose terms and formulae into their atomic parts. The following definitions now make use of this concept

(i) **sub-formulae**
For any formula $\varphi \in \text{form}(\mathcal{L})$ define the set $\text{sub}(\varphi) \subseteq \text{form}(\mathcal{L})$ of all **sub-formulae** of φ by inverse recursion

$$\begin{aligned}
\text{sub}(\varphi) &:= \{\varphi\} \text{ if } \varphi \text{ is atomic} \\
\text{sub}(\neg\varphi) &:= \{(\neg\varphi)\} \cup \text{sub}(\varphi) \\
\text{sub}(\varphi \to \psi) &:= \{(\varphi \to \psi)\} \cup \text{sub}(\varphi) \cup \text{sub}(\psi) \\
\text{sub}(\exists_i\, x\, \varphi) &:= \{(\exists_i\, x\, \varphi)\} \cup \text{sub}(\varphi) \\
\text{sub}(\forall_i\, x\, \varphi) &:= \{(\forall_i\, x\, \varphi)\} \cup \text{sub}(\varphi)
\end{aligned}$$

For any two formulae φ, $\psi \in \text{form}(\mathcal{L})$ we say that φ is a *sub-formula* of ψ, which we will abbreviate by $\varphi \leq \psi$, iff $\varphi \in \text{sub}(\psi)$.

(ii) **free variables**
Let $a \in \mathcal{L}$ be a term or formula of $\mathcal{L}$, then we next define the set of **free variables** of a by inverse recursion

$$\begin{aligned}
\text{free}(x) &:= \{x\} \\
\text{free}(c) &:= \emptyset \\
\text{free}(f t_1 \ldots t_k) &:= \text{free}(t_1) \cup \cdots \cup \text{free}(t_k) \\
\text{free}(R t_1 \ldots t_k) &:= \text{free}(t_1) \cup \cdots \cup \text{free}(t_k) \\
\text{free}(s =_i t) &:= \text{free}(s) \cup \text{free}(t) \\
\text{free}(\neg\varphi) &:= \text{free}(\varphi) \\
\text{free}(\varphi \to \psi) &:= \text{free}(\varphi) \cup \text{free}(\psi) \\
\text{free}(\exists_i\, x\, \varphi) &:= \text{free}(\varphi) \setminus \{x\} \\
\text{free}(\forall_i\, x\, \varphi) &:= \text{free}(\varphi) \setminus \{x\}
\end{aligned}$$

where we used the same notations as in definition 3 and the sorts of all terms are supposed to be adequate. To express that φ is a formula having the free variables $\text{free}(\varphi) = \{\, x_1, \ldots, x_k \,\}$ we will occasionally write $\varphi(x_1, \ldots, x_k)$. Analogously we will write $\varphi(x_1, \ldots, x_k, \ldots)$ in the case that $x_1, \ldots, x_k$ are some (but not necessarily all) free variables of φ, that is to say, when $\{\, x_1, \ldots, x_k \,\} \subseteq \text{free}(\varphi)$.

(iii) **sentences**
A formula φ of $\mathcal{L}$ is called to be a **sentence**, iff it does not contain any free variables and we denote the set of all such by

$$\mathrm{sen}(\mathcal{L}) \;:=\; \{\, \varphi \in \mathrm{form}(\mathcal{L}) \mid \mathrm{free}(\varphi) = \emptyset \,\}$$

(iv) **free occurance**
If (for $k \in 1 \ldots \ell(\varphi)$) $\varphi[k] \in \mathrm{var}(\mathcal{L})$ is a variable symbol $x := \varphi[k]$ then we say that this is a **bound occurance** of x in φ, iff at this place x is quantified over. That is $\varphi = \alpha\varphi'\beta$ where (1) $\varphi' \leq \varphi$ is a subformula of the form $\varphi' = Qx\,\varphi''$ for some quantifier $Q \in \mathrm{quant}(\mathcal{L})$ and (2) the location $\varphi[k]$ is part of φ''. If $\varphi[k] \in \mathrm{var}(\mathcal{L})$ is a variable $x = \varphi[k]$, that is *not* quantified over, we say this is a **free occurance** of x in φ.

(v) **freely substitutable**
Let $x \in \mathrm{var}(\mathcal{L})$ be a variable symbol, $t \in \mathrm{term}(\mathcal{L})$ be a term and $\varphi \in \mathrm{form}(\mathcal{L})$ be a formula of $\mathcal{L}$. Then we say, that x is **freely substitutable** by t, iff $\mathrm{sort}(x) = \mathrm{sort}(t)$ and at any free occurance of x in φ the variable x is not part of some sub-formula of the form $Qy\,\varphi'' \leq \varphi$ where $Q \in \mathrm{quant}(\mathcal{L})$ is quantifier and $y \in \mathrm{free}(t)$ also is a free variable of t. More formally that is:

$$\mathrm{free}(t) \cap \left\{ y \in \mathrm{var}(\mathcal{L}) \;\middle|\; \begin{array}{ll} \varphi' \leq \varphi & \text{sub-formula} \\ \text{such that} & \varphi' = Qy\,\varphi'' \\ \text{and} & x \in \mathrm{free}(\varphi') \end{array} \right\} \;=\; \emptyset$$

(vi) **substitution**
Let $x \in \mathrm{var}(\mathcal{L})$ be a variable symbol and $t \in \mathrm{term}(\mathcal{L})$ be a term of the same sort, $\mathrm{sort}(x) = \mathrm{sort}(t)$, and $a \in \mathcal{L}$ be a term or formula. We will now define the notion of **substituting** x by t in a using inverse recursion. Thus we start with $\ell(a) = 1$, in this case we let

$$a[x:t] := \begin{cases} a & \text{for } a \neq x \\ t & \text{for } a = x \end{cases}$$

If now $2 \leq \ell(a)$ then we continue (here $Q \in \mathrm{quant}(\mathcal{L})$ is any quantifier and $y \in \mathrm{var}(\mathcal{L})$ is any variable symbol of $\mathcal{L}$) by letting

$$a[x:t] := \begin{cases} f t_1[x:t] \ldots t_k[x:t] & \text{for } a = f t_1 \ldots t_k \\ R t_1[x:t] \ldots t_k[x:t] & \text{for } a = R t_1 \ldots t_k \\ (t_1[x:t] =_i t_2[x:t]) & \text{for } a = (t_1 =_i t_2) \\ (\neg\varphi[x:t]) & \text{for } a = (\neg\varphi) \\ (\varphi_1[x:t] \to \varphi_2[x:t]) & \text{for } a = (\varphi_1 \to \varphi_2) \\ Q\,y\,\varphi[x:t] & \text{for } a = Q\,y\,\varphi \text{ and } x \neq y \\ Q\,y\,\varphi & \text{for } a = Q\,y\,\varphi \text{ and } x = y \end{cases}$$

Example 6:
So what is this *freely substitutable* business all about? Why care? As a first example let us take a look at the formula $\varphi = \exists\, y\,(y = x)$. It has the free variable x only. So what happens if we replace x with a third variable z?

$$\varphi[x : z] \;\; = \;\; \exists\, y\,\big(y = z\big)$$

Nothing actually, the logical content of $\varphi[x : z]$ is precisely the same, both formulae express a truism, as we could take $y = x$ or $y = z$ respectively. Now we replace x by the term $t = y + z$, in this case we get

$$\varphi[x : (x + y)] \;\; = \;\; \exists\, y\,\big(y = y + z\big)$$

In presence of the group axioms this would be $\exists\, y(0 = z)$ So the truth of the statement suddenly depends on the content of z. For $z = 0$ it would be true, but false in any other case. So what happened? We replaced x in a position where the formula quantifies over the variable y, that also occurs in the term t by which we substituted. This is precisely the situation that we exclude by demanding that x shall be freely substitutable by t.

As a second example consider $\varphi = \forall\, u\,((u = v + x) \wedge \forall\, v\,(v = u + x))$. Let us first mark any free appearance of the variables u, v and x in the formula φ by placing a dot on top of the respective symbol

$$\varphi \;\; = \;\; \forall\, u\,\big((u = \dot{v} + \dot{x}) \wedge \forall\, v\,(v = u + \dot{x})\big)$$

- In φ the free variable x *is not* freely substitutable by the term $t := u$, as $\varphi' := (u = v + x) \wedge \forall\, v\,(v = u + x)$ is a sub-formula, with free variable $x \neq u$ such that $\varphi = \forall\, u\,\varphi'$ and u is a variable of t.
- In φ the free variable x *is not* freely substitutable by the term $t := v+x$, as $\varphi' := (v = u + x)$ is a sub-formula with free variable $x \neq v$ such that $\forall\, v\,\varphi' \leq \varphi$ and v is a variable of t.
- In φ the free variable x *is* freely substitutable by the term $t := x + x$ as the quantifiers in φ do not meet the variable x of the term t.
- In φ the free variable v *is* freely substitutable by the term $t := x$ as in the *one* free appearance of v in φ the quantifier $\forall\, u$ does not bind a variable of t. In fact, by definition, the substitution leaves the right part of the formula unchanged

$$\varphi[v : x] \;\; = \;\; \forall\, u\,\big((u = x + x) \wedge \forall\, v\,(v = u + x)\big)$$

3 Examples of Many-Sorted Languages

Example 7:
Let us begin with a neat little example: plane geometry. Here we have two sorts of objects - points and lines - which we represent by two sorts $I = \{ p, \ell \}$. If the point P lies on the line L we would like to write $P \in L$. For lines we have the operation of the intersection $\cap$ and the relation $\parallel$ of being parallel. Altogether that is

$$\begin{aligned} \mathcal{C} &= \emptyset \\ \mathcal{F} &= \{ \cap \} \\ \mathcal{R} &= \{ \in, \parallel \} \end{aligned}$$

The sorts are clear: First of all $\parallel$ is a relation between lines, therefore $\text{sort}(\parallel) = (\ell, \ell)$. Now $P \in L$ shall indicate that P lies on L, hence $\in$ has to have the $\text{sort}(\in) = (p, \ell)$. And as $\cap$ turns two lines into a point we have $\text{sort}(\cap) = (\ell, \ell, p)$. Let us now formulate that any two lines, that are not parallel, intersect in a point:

$$\forall_\ell K \, \forall_\ell L \left(\neg (K \parallel L) \to \exists_p P \left(K \cap L =_p P \right) \right)$$

NOTA we already used the standard notation $K \cap P := \cap(K, L)$ here to increase the readability of the formula. And we will continue to do so in the following examples with all binary operations like $+$, $-$, $\cdot$ and $:$.

If we want to add a far point, as in projective geometry, we can do so as a constant symbol, say $\mathcal{C} = \{ \infty \}$ of the $\text{sort}(\infty) = p$. Using this we could formulate, that any two distinct, parallel lines intersect in that point:

$$\forall_\ell K \, \forall_\ell L \left(\left((K \parallel L) \wedge \neg (K =_\ell L) \right) \to (K \cap L =_p \infty) \right)$$

Example 8:
Let us continue with a one-sorted example: The language $\mathcal{L}_r$ of rings. To be one-sorted means $I = \{ r \}$ contains a single element only. So what do we need for rings? The constants 0 and 1, the addition $+$ and multiplication $\cdot$ are a minimum. So we start with

$$\begin{aligned} \mathcal{C} &= \{ 0_r, 1_r \} \\ \mathcal{F} &= \{ +_r, \cdot_r \} \\ \mathcal{R} &= \emptyset \end{aligned}$$

As we only have one sort r the constants are of this sort of course $\text{sort}(0_r) = r$ and $\text{sort}(1_r) = r$. And as $+$ and $\cdot$ are binary operations we find their sort to be $\text{sort}(+_r) = (r, r, r)$ and $\text{sort}(\cdot_r) = (r, r, r)$.

If we want to prove a formula in this language, for any ring, we need to start with the properties of rings. These are sentences in this language. First of all a ring is a commutative group under $+$ with neutral element 0, that is

$$\begin{array}{lll}
\text{(G1)} & \forall_r a\, \forall_r b\, \forall_r c & a +_r (b +_r c) \;=_r\; (a +_r b) +_r c \\
\text{(G2)} & \forall_r a\, \forall_r b & a +_r b \;=_r\; b +_r a \\
\text{(G3)} & \forall_r a & a +_r 0_r \;=_r\; a \\
\text{(G4)} & \forall_r a\, \exists_r b & a +_r b \;=_r\; 0_r
\end{array}$$

The next set of formulae expresses that a ring is a commutative monoid under $\cdot$ with neutral element 1 and that $+$ and $\cdot$ are interlocked by the law of distributivity. If (G1) to (G4) and (R1) to (R4) are satisfied, we have a *commutative ring* (with unity)

$$\begin{array}{lll}
\text{(R1)} & \forall_r a\, \forall_r b\, \forall_r c & a \cdot_r (b \cdot_r c) \;=_r\; (a \cdot_r b) \cdot_r c \\
\text{(R2)} & \forall_r a\, \forall_r b & a \cdot_r b \;=_r\; b \cdot_r a \\
\text{(R3)} & \forall_r a & a \cdot_r 1_r \;=_r\; a \\
\text{(R4)} & \forall_r a\, \forall_r b\, \forall_r c & a \cdot_r (b +_r c) \;=_r\; (a \cdot_r b) +_r (a \cdot_r c)
\end{array}$$

As the addition $+$ has an inverse, the subtraction $-$ it can be helpful to also introduce this to our language, that is we even take $\mathcal{F} = \{ +, -, \cdot \}$ where $\mathrm{sort}(-) = (r, r, r)$, as well. We will call this expanded language the *language of rings* and denote it by

$$\mathcal{L}_r \;:=\; \mathcal{L}_{\{r\}}\big(\{ 0_r, 1_r \}, \{ +_r, -_r, \cdot_r \}, \emptyset, \mathrm{sort} \big)$$

Picking up this function symbol we need to fix the meaning of $-$ as the inverse of $+$. So in this case we need another property, namely

$$\text{(AI)} \quad \forall_r a\, \forall_r b\, \forall_r c \quad (c -_r b =_r a) \;\leftrightarrow\; (c =_r a +_r b)$$

We will later use the *theory of integral* domains. This is the set T_{id} of sentences (G1) to (G4), (R1), ..., (R4), (AI) and (ID) where

$$\text{(ID)} \quad \forall_r a\, \forall_r b\, \forall_r c \quad (a \cdot_r b =_r 0) \;\rightarrow\; (a =_r 0 \,\vee\, b =_r 0)$$

The only property that is missing for a field is: Any $a \neq 0$ has a multiplicative inverse, that is $\forall_r a (\neg(a =_r 0_r) \rightarrow \exists_r b\, a \cdot_r b =_r 1_r)$. This gives rise to a division function $:_r$ of $\mathrm{sort}(:_r) = (r, r, r)$ that we include for the *language of fields* (where we rename the sort from r to f)

$$\mathcal{L}_f \;:=\; \mathcal{L}_{\{f\}}\big(\{ 0_f, 1_f \}, \{ +_f, -_f, \cdot_f, :_f \}, \emptyset, \mathrm{sort} \big)$$

We fix the meaning of $:$ as the inverse of $\cdot$ so for a field we require the properties (G1) to (G4), (R1) to (R4), (AI) and (MI) where

$$\text{(MI)} \quad \forall_f a\, \forall_f b\, \forall_f c \quad \Big(\neg(a =_f 0_f) \rightarrow \big((c :_f b =_f a) \leftrightarrow (c =_f a \cdot_f b) \big) \Big)$$

We will soon consider *algebraically closed* fields. In order to express that the variables of $\mathcal{L}_f$ belong to an algebraically closed field we need a whole scheme of axioms: For each $1 \leq n \in \mathbb{N}$ pick up the statement

$$(\mathrm{AC}_n) \quad \forall_f a_0 \ \ldots \ \forall_f a_n \ \exists_f z \quad (a_n \cdot_f z^n) +_f \cdots +_f (a_1 \cdot_f z) +_f a_0 =_f 0_f$$

where we already used the abbreviation $z^n := z \cdot_f z \cdot_f \cdots \cdot_f z$ (n-times). Then the *theory of algebraically closed fields* T_{ac} consists of the statements (G1) to (G4), (R1) to (R4), (AI), (MI) and (AC_n) for any $1 \leq n \in \mathbb{N}$.

Example 9:
Our next example addresses a basic notion of linear algebra: Modules over (commutative) rings. Likewise we could append the properties for the base ring to be a field to have the notion of a vector-space. Clearly this language requires two sorts $I = \{ r, m \}$, one for the base ring r, the other for the module m. The ring has the constants 0_r and 1_r, the module 0_m only. There are no relations but the addition $+_r$ and multiplication $\cdot_r$ of scalars, the addition $+_m$ of vectors and a scalar multiplication $\cdot_s$. Altogether this is

$$\begin{aligned} \mathcal{C} &= \{ 0_r, 1_r, 0_m \} \\ \mathcal{F} &= \{ +_r, \cdot_r, +_m, \cdot_s \} \\ \mathcal{R} &= \emptyset \end{aligned}$$

The assignment of sorts is obvious, resp. well-known here, but it has to be given beforehand: $\mathrm{sort}(0_r) = r$, $\mathrm{sort}(1_r) = r$, $\mathrm{sort}(0_m) = m$, continued by $\mathrm{sort}(+_r) = (r, r, r)$, $\mathrm{sort}(\cdot_r) = (r, r, r)$, $\mathrm{sort}(+_m) = (m, m, m)$ and finally $\mathrm{sort}(\cdot_s) = (r, m, m)$. So much for the definition of the language, let's see what we can do with it: We can literally repeat (G1) to (G4) and (R1) to (R4) to express that the variables of sort r belong to a commutative ring. Next the variables of sort m belong to a (left) module over this ring. That is they form a commutative group under $+_m$ with neutral element 0_m. It is easy to repeat these properties in terms of formluae

$$\begin{array}{lcll} (\mathrm{M1}) & \forall_m x \, \forall_m y \, \forall_m z & x +_m (y +_m z) & =_m \ (x +_m y) +_m z \\ (\mathrm{M2}) & \forall_m x \, \forall_m y & x +_m y & =_m \ y +_m x \\ (\mathrm{M3}) & \forall_m x & x +_m 0_m & =_m \ x \\ (\mathrm{M4}) & \forall_m x \, \exists_m y & x +_m y & =_m \ 0_m \end{array}$$

It takes another 4 formula to express that $\cdot_s$ truly is a scalar multiplication between the variables of sort r and sort m. In fact these are compatibility conditions between all the 4 operations involved

$$\begin{array}{lcll} (\mathrm{S1}) & \forall_r a \, \forall_m x \, \forall_m y & a \cdot_s (x +_m y) & =_m \ (a \cdot_s x) +_m (a \cdot_s y) \\ (\mathrm{S2}) & \forall_r a \, \forall_r b \, \forall_m x & (a +_r b) \cdot_s x & =_m \ (a \cdot_s x) +_m (b \cdot_s x) \\ (\mathrm{S3}) & \forall_r a \, \forall_r b \, \forall_m x & (a \cdot_r b) \cdot_s x & =_m \ a \cdot_s (b \cdot_s x) \\ (\mathrm{S4}) & \forall_m x & 1_r \cdot_s x & =_m \ x \end{array}$$

Note that all these formulae are sentences - all the free variables have been quantified over. This is commonplace for sets of axioms - what use would it have to employ variables just as variables? Also we see that substituting is a completely natural thing to do, e.g. regard the formula $\varphi(x) = \exists_m z : x +_m z =_m 0_m$, then we might want to replace the variable x by the term $t = (x +_m y)$. Doing this we end up with $\psi(x, y) := \varphi[x : t] = \exists_m z : (x +_m y) + z =_m 0_m$ that now has two free variables, x and y.

Example 10:
In the next example let us introduce the language of ring extensions $S : R$ - this is a language containing two sorts $I = \{ r, s \}$, one for the subring R, the other for the larger ring S. It obviously has the constants and functions (but no relations again)

$$\begin{aligned} \mathcal{C} &= \{ 0_r, 1_r, 0_s, 1_s \} \\ \mathcal{R} &= \emptyset \\ \mathcal{F} &= \{ +_r, \cdot_r, +_s, \cdot_s, \iota \} \end{aligned}$$

The assignment of sorts is the same as in example 9, with the exception of ι, which is a 1-ary function of the sort $\mathrm{sort}(\iota) = (r, s)$. We will explain the role of ι below.

So much for the language, to express that both R and S are commutative rings, we would have to repeat the respective 2 times 8 formulae (G1) to (R4) of example 8. However then we've just got two (commutative) rings (with identity) and we haven't said a word about the fact that R is contained in S. To do this we included the function symbol ι: The following formulae express that ι truly embeds R into S

$$\begin{array}{lll} \text{(E1)} & \forall_r a \, \forall_r b & \iota(a +_r b) \;=_s\; \iota(a) +_s \iota(b) \\ \text{(E2)} & \forall_r a \, \forall_r b & \iota(a \cdot_r b) \;=_s\; \iota(a) \cdot_s \iota(b) \\ \text{(E3)} & \forall_r a \, \forall_r b & a \;=_r\; b \leftrightarrow \iota(a) \;=_s\; \iota(b) \\ \text{(E4)} & & \iota(1_r) \;=_s\; 1_s \end{array}$$

This example demonstrates how we can evade the disjointness of the sorts that this formalism would otherwise yield.

Example 11:
As a final example let us present the language of modern set theory. It obviously contains two sorts of variables again $I = \{ s, c \}$ which we interpret as sets and classes respectively. It contains a constant, two relations and a function that takes sets to classes:

$$\begin{aligned} \mathcal{C} &= \{ \emptyset \} \\ \mathcal{R} &= \{ \in_s, \in_c \} \\ \mathcal{F} &= \{ \iota \} \end{aligned}$$

Hereby we assign $\text{sort}(\emptyset) = s$, $\text{sort}(\in_s) = (s, s)$, $\text{sort}(\in_c) = (s, c)$ and as before $\text{sort}(\iota) = (s, c)$. Note that by this construction it is already forbidden to speak of a class that is element of another class - sets and classes can only contain sets. The *axiom of extensionality* is a formula that expresses, that any two classes B and C are equal if and only if they share the same sets x as elements:

$$\forall_c B \, \forall_c C \; : \; \big(B =_c C \leftrightarrow \forall_s x \, (x \in_c B \leftrightarrow x \in_c C)\big)$$

A *small* class is a class S that also is a set. In terms of the embedding ι this can very simply be expressed in the following formula

$$\exists_s X \; : \; S =_c \iota(X)$$

NOTA that many-sorted languages can even mimic the workings of second-order languages in which it is allowed to quantify over relation and function symbols (not only over variable symbols) by introducing a sort for the relation symbols and another for the function symbols. But with the introduction of the language $\mathcal{L}(\in)$ of sets, second-order languages have become next to pointless already, so we will not pursue this path any further.

It is tempting to write out an universal language, that can cover all of algebra, maybe all of mathematics. Well, the language of sets and classes is such a universal language in some sense, but this is highly non-specific. So wouldn't it be nice to have a language for all of algebra, for example? The answer however is: no!

Mathematics already has a universal language - the metalanguage of naive logic and set theory, refined by their axioms in formalized form. What formal logic provides is another tool how to denote and prove theorems in any other field of mathematics. Instead of prescribing how proofs are supposed to be, we should rather look at formal logic as an asset that opens up alternative ways of proving theorems. It should not be seen as an alternative, but rather as an enhancement.

And to do so we do not need the one universal language, but rather several, lean languages specifically tailored for the problem at hand. Without going into details we want to give an example of what formal logic can accomplish: quantifier elimination.

The first example of quantifier elimination is the determinant. Consider a square matrix A over a field F. It is well known, that A is invertible if and only if its determinant is non-zero. That is we have the equivalent statements

(a) $\exists B : (AB = \mathbb{1}_n) \wedge (BA = \mathbb{1}_n)$

(b) $\neg(\det(A) = 0)$

The advantage is clear: In (a) we would have to check an infinite number of possible inverse matrices B, whereas in (b) we can simply perform a finite computation. So if we can eliminate the quantifiers from a formula there is a finite way to determine its truth. To show off, let us provide some advanced results, without proof [that are already formulated using the notions we are about to introduce in the following sections]:

Definition 12:

(i) Let $\mathcal{L} = \mathcal{L}_I(\mathcal{C}, \mathcal{F}, \mathcal{R}, \text{sort})$ be any many-sorted, formal language and let $\eta \in \text{form}(\mathcal{L})$ be a formula therein. The we say that η is **quantifier-free**, if it can be generated, as a formula, by rules (F1) and (F2) alone.

(ii) A theory T (that is a set of sentences $T \subseteq \text{sen}(\mathcal{L})$) is said to **admit quantifier-elimination**, iff for any formula $\varphi \in \text{form}(\mathcal{L})$ there is another formula $\eta \in \text{form}(\mathcal{L})$ such that

(1) η is quantifier-free, and

(2) $T \vdash (\varphi \leftrightarrow \eta)$

(iii) If $(X, \varrho) \in \text{real}(\mathcal{L})$ is a realization of $\mathcal{L}$, then we say that (X, ϱ) **admits quantifier-elimination**, iff its theory $\text{th}(X, \varrho)$ admits quantifier elimination, where we define

$$\text{th}(X, \varrho) \;:=\; \{\, \varphi \in \text{sen}(\mathcal{L}) \mid (X, \varrho) \models \varphi \,\}$$

Theorem 13:

(i) Consider the language $\mathcal{L}_f$ of fields introduced in 8, then the theory T_{ac} of algebraically closed fields admits quantifier elimination.

(ii) Consider the language $\mathcal{L}_r$ of rings introduced in 8 and let R be an integral domain (i.e. a realization of $\mathcal{L}_r$ that satisfies $R \models T_{id}$). Then the following statements are equivalent

(a) R admits quantifier-elimination

(b) R is finite, or an algebraically closed field

NOTA a theory T admits quantifier elimination iff it is *substructure-complete* [Felgner 3.1]. Then it can be shown (introducing the *amalgation property*) [Felgner 3.2] that T_{ac} is substructure-complete [Felgner 3.4]. An algorithm for quantifier-elimination in algebraically closed fields is presented in [KrKr]. The proof of (ii) is a beautiful synthesis of algebraic arguments supplemented by the benefits of quantifier-elimination [Felgner 8.11].

Thank You for previewing this eBook

You can read the full version of this eBook in different formats:

- HTML (Free /Available to everyone)
- PDF / TXT (Available to V.I.P. members. Free Standard members can access up to 5 PDF/TXT eBooks per month each month)
- Epub & Mobipocket (Exclusive to V.I.P. members)

To download this full book, simply select the format you desire below