A Categorical Approach to Logics and Logic Homomorphisms

Robert Helgesson
`c03rhn@cs.umu.se`

February 9, 2007
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Patrik Eklund
Examiner: Per Lindström

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN

Abstract

This master's thesis presents a number of important concepts in logic such as models, entailment, and proof calculi within the framework of category theory. By describing these concepts as categories, a tremendous amount of generality and power is gained. In particular, this approach makes it possible to reason about maps from one logic to another in a consistent and convenient manner. By a consistent map is meant that the *truth* stays invariant, that is, a statement true in the source logic is mapped to a similarly true statement in the target logic. Conversely, a statement false in the source logic is mapped to a statement false in the target logic.

While the thesis focuses on the theoretical notions outlined above, a brief coverage of two practical applications is given as a means to illustrate the utility of these notions. Concluding the text is a chapter containing a discussion and a section wherein possible future work is presented.

In an effort to make the text mostly self-contained, concepts beyond basic discrete mathematics are duly introduced with definitions and examples. These include, for example, category theory and universal algebra, both of which are essential in this description of logic. In other words, little prior knowledge about the subject is assumed.

Contents

List of Notation

$A = (A_i)_{i\in I}$	A family of sets	2
$\mathtt{C}$	A category	6
$\mathrm{Ob}(\mathtt{C})$	The set of objects in the category $\mathtt{C}$	6
$\mathrm{Hom}_{\mathtt{C}}(A,B)$	The set of $\mathtt{C}$-morphisms having domain A and codomain B	6
$\mathtt{C}^{op}$	The opposite (or dual) category of the category $\mathtt{C}$	8
$\mathcal{P}$	The powerset functor	12
$\Sigma = (S,\Omega)$	A many-sorted equational signature	18
$\mathrm{Op}(\Sigma)$	The set of operation names of a signature Σ	18
μ	A signature morphism	21
$\mathtt{Sign}^{Eq}$	Category of equational signatures	22
$T_\Sigma X$	The family of Σ-terms using variables in X	23
$\mathrm{var}(t)$	The family of variables for a term t	23
$\mathfrak{v}$	A variable assignment function	27
$\mathtt{Alg}_\Sigma$	Category of Σ-algebras	30
$\mathscr{I}$	An institution	32
$\mathtt{Sign}$	The category of signatures	32
Sen	Sentence functor	32
Mod	Model functor	32
$\models$	Model-theoretic family of satisfaction relations	32
$\vDash$	Axiomatic family of satisfaction relations	33
$\mathrm{free}(\varphi)$	The family of free variables in a formula φ	40
$\mathscr{E}$	An entailment system	43
$\mathscr{L}$	A logic	45
$\mathcal{S}$	The functor extracting from a theory its signature	48
$\mathcal{A}$	The functor extracting from a theory its set of axioms	48
$\mathcal{T}$	The functor expanding a signature to an axiom-free theory	48
(Φ,α)	Map of entailment systems	50
$\mathtt{Ent}$	The category of entailment systems	53
(Φ,α,β)	Map of institutions	53

Chapter 1

Introduction

The concept of mathematical logic certainly has a long and lustrous history. From its dawn primarily in ancient Greece to its development into a field of modern rigorous study in the 1800's, it has formed a foundation on which many great works of mathematics and computing science have been constructed [7]. As a result of the attention paid to the topic, there exists today not a single all-encompassing notion of logic but a wide variety of logics for all imaginable purposes. The situation has been succinctly summarized by Joseph Goguen and Rod Burstall who in *Introducing Institutions* [16] wrote the—in the field—well known quote:

> There is a population explosion among the logical systems being used in computer science. Examples include first order logic (with and without equality), equational logic, Horn clause logic, second order logic, higher order logic, infinitary logic, dynamic logic, process logic, temporal logic, and modal logic; moreover, there is a tendency for each theorem prover to have its own idiosyncratic logical system.

With all these possible logics, it is of interest to study the relationships that exists—or does not exist—between logics. This thesis forms an attempt to, in an understandable but still rigorous manner, present a framework which allows the description of many logics. This presentation is based on the extensive works of not only the aforementioned Joseph Goguen and Rod Burstall but also José Meseguer and many other skillful mathematicians and computing scientists.

The framework of logics mentioned above will be presented using the notions introduced by so-called category theory and universal algebra. Briefly stated, category theory is a field of mathematics which allow the study of mathematical structures and transformations between them in a consistent and convenient manner. Similarly, universal algebra is the mathematical field in which one study abstract algebras and notions common to them. These topics—and others covered in this text—will be treated in a fairly rigorous manner and as a result, may be thought of as an introduction to the subject for those previously unfamiliar of it. To this end, the material is presented with an emphasis placed on clarity and accessibility. By this is meant that the intent is to make the text relatively self-contained and all required knowledge will be presented in an incremental fashion. More specifically, no prior familiarity in the area of category theory or universal algebra is assumed. Prior knowledge of the basic notions of discrete mathematics and mathematical logic is, however, assumed. For some easily forgotten details, brief descriptions have been included though. If needed, the required knowledge may be recalled in most texts providing an introduction to basic discrete mathematics and algebra, see e.g. [21]. Concepts and notations outside the scope of basic discrete mathematics will be thoroughly described through formal discussions and both conceptual as well as concrete examples. Unfortunately this makes the thesis very "definition heavy" but hopefully not more so than what the patient reader is able to accept.

It should be pointed out that the examples given in the text will in general not be very complex; this is an intentional decision. There are two primary reasons underlying this decision: One reason is, as can be imagined, the problem of space and time. In short, the addition of more complex examples would have meant the removal of other content—finding the right balance between depth and breadth has been a considerable challenge. The other reason behind choosing less complex examples is one of keeping a general nature of the theories rather than specific problem instances. There is always a risk that adding complex examples will force the reader into a quagmire of gritty details which are not directly related to the point being made. For this reason, smaller—but still non-trivial—examples which clearly show the mechanics of the notions presented have been favored.

The overall structure of this document is such that

Chapter 2 covers the basic notions of category theory;

Chapter 3 presents universal algebra within the framework of category theory;

Chapter 4 is the main focus of the thesis and contains thorough descriptions of many notions of logic from a categorical viewpoint;

Chapter 5 discusses, on a less rigorous level, a few possible applications of the concepts introduced throughout the text; and

Chapter 6 contains a brief discussion of the topics covered as well as some conclusions and future work.

Closing this introductory chapter is a brief coverage of a basic—but crucial—mathematical notion which will be used extensively throughout this text. This notion is the one of *families of sets*—also commonly referred to as a *collections of sets* or *indexed sets*. The description here will by no means be a fully formal definition but should give the reader an intuitive sense of what a family of sets is.

Definition 1 (Families of sets) Let $A = (A_i)_{i \in I}$ be a family of sets *indexed* by some, potentially unbounded, set I. We will, as a convenient notation, assume the convention that $A = \{\{a_1, a_2, \ldots\}_1, \{b_1, b_2, \ldots\}_2, \ldots\}$ is the family of sets A such that $A_1 = \{a_1, a_2, \ldots\}, A_2 = \{b_1, b_2, \ldots\}, \ldots$.

We extend the idea of a family of sets to a *family of functions* by stating that, if A and B are two families of sets indexed by I, then $f : A \longrightarrow B$ is a family of functions containing a function $f_i : A_i \longrightarrow B_i$ for each $i \in I$. Composition of two families of I-indexed functions $f : A \longrightarrow B, g : B \longrightarrow C$, is yet another family of functions $f \circ g : A \longrightarrow C$ defined in terms of function composition such that $f_i \circ g_i : A_i \longrightarrow C_i$ for each $i \in I$. We extend the notions of membership by stating that $x \in A$ if $x \in A_i$ for some $i \in I$. Other set theoretic operations extend in a similar, natural, fashion. For example, the union C of two I-indexed families of sets A, B is $C = A \cup B$ where $C_i = A_i \cup B_i$ for each $i \in I$.

Finally, as this is a computing scientific text the set of all natural numbers, $\mathbb{N}$, will be assumed to contain the number zero, that is, $\mathbb{N} = \{0, 1, 2, \ldots\}$.

This brief recollection of basic mathematical notions should be sufficient for most to fully comprehend this text. Of course, quite a few new concepts—and their accompanying notation—will be introduced in the chapters to come. But, as previously mentioned, this will be done in an incremental fashion and little difficulties should therefore be posed for the diligent and interested reader.

A lot of ground will be covered in the chapters to come so without further ado; let us take the plunge!

Chapter 2

Category Theory

From its first embryonic developments by Eilenberg and Mac Lane in the first half of the 1940's—with their seminal article *General theory of natural equivalences* published in 1945, see [9], being the major introduction—category theory has developed into a field of mathematical study in its own right. Then, certain features of category theory known as functors and natural transformations—described in Sections 2.2 and 2.3, respectively—were developed as a means to solve problems related to group theory and topology[1], in particular one was concerned about how transitions among mathematical structures could be understood. Today, category-theoretic notions have been applied in many different areas of mathematics and computing science and, as a result, a large number of elegant results have been produced. It has even been a useful tool in the study of the foundations of mathematics, more specifically it may be considered an alternative to axiomatic set theory as the basis of our understanding of mathematics. The work of Lawvere has been particularly prominent in this area [24, 2, 3].

Within the field of theoretical computing science, category theory is able to provide real benefits. The case of term unification is a good example of this: First, the correctness proof of the traditional unification algorithm may be greatly simplified when approached from a categorical point of view. Second, the notion of what unification can be broadened in an elegant manner from the simple term unification to also be applicable on differential equations and type inference [14]. In his article *A Categorical Manifesto* [15], Goguen presents a good overview of the potential benefits—and pitfalls—of applying category theory within the field of theoretical computing science.

The notions of category theory have not only been applied in the strictly theoretical area of computing science, it has also proven itself quite useful in practical applications. As an example of this, one might mention the elegant way in which the idea of *monads* has allowed pure functional languages—such as Haskell—to encapsulate and separate effect-full computations without compromising purity. See e.g. [32, 43].

While monads will not be discussed to any great degree in this text, we will in this chapter present other basic categorical notions together with both concrete and more abstract examples. This presentation will form a foundation for the chapters that follow. However, since the presentation is kept at a general level, it may also be considered as a brief introduction to basic category theory in general. The various definitions and notations found here are adapted primarily from the well known books by Barr and Wells [2]; Adámek, Herrlich, and Strecker [1]; as well as Mac Lane [28].

[1]Since this text will not consider these structures further, the interested reader is referred to e.g. [22, 8] for more in depth information.

2.1 Categories

Informally a category provides a convenient, yet extremely flexible, way to describe mathematical structures and, in particular, how these structures relate to each other. To make this statement more concrete, a number of examples will be given, but to do so, we first need the formal definition of a category.

Definition 2 (Categories) A *category* $\mathtt{C}$ consists of

- a collection of $\mathtt{C}$-*objects*, represented by the set $\mathrm{Ob}(\mathtt{C})$, and
- for each pair of $\mathtt{C}$-objects A, B, a collection of $\mathtt{C}$-*morphisms* from A to B, represented by the set $\mathrm{Hom}_{\mathtt{C}}(A,B)$ or simply $\mathrm{Hom}(A,B)$ if the intended category is either obvious from context or irrelevant to the argument.

For simplicity, whenever context allows, $\mathtt{C}$-objects and $\mathtt{C}$-morphisms will be called just *objects* and *morphisms*, respectively.

A morphism $f \in \mathrm{Hom}(A,B)$ will typically be denoted $A \xrightarrow{f} B$ or $f : A \longrightarrow B$. Additionally, A is said to be the *domain* of f and similarly, B is said to be its *codomain*. Finally, to complete the definition, if $A \xrightarrow{f} B$, $B \xrightarrow{g} C$, and $C \xrightarrow{h} D$ are arbitrary morphisms in $\mathtt{C}$ then

(i) there is a composition operation, $\circ$, and $A \xrightarrow{g\circ f} C$ is a morphism in $\mathtt{C}$;

(ii) composition is associative, that is,

$$h\circ(g\circ f) = (h\circ g)\circ f; \text{ and}$$

(iii) there exists *identity morphisms* $\mathrm{id}_A \in \mathrm{Hom}(A,A)$ and $\mathrm{id}_B \in \mathrm{Hom}(B,B)$ such that

$$f\circ \mathrm{id}_A = f \text{ and } \mathrm{id}_B \circ f = f.$$

□

Remark 1 From the definition it may appear as if morphisms are the same as functions and while they in particular cases—like in the example that follows—can be equivalent, this is not true in general. Far from it in fact; many of the morphisms which will be presented in this thesis are not functions. This connection with functions is an underlying reason behind the notation used for morphism composition. Since this notation may appear counter intuitive some authors prefer to write $f;g$ over $g\circ f$. □

Remark 2 In general, instead of writing $A \xrightarrow{f} B$, $B \xrightarrow{g} C$, and $C \xrightarrow{h} D$, we may simplify somewhat and just write $A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D$. □

Remark 3 Note that in other texts the concept of a *morphism* may instead be referred to as an *arrow*. Also, the identity morphism is in some sources denoted by the digit 1, that is, for some object A, $\mathrm{id}_A = 1_A$. Although this notation will not be used here it is so wide spread that it is well worth remembering. □

Using the definition of a category, it is beneficial to make the concept more concrete using a few examples. These are only a very small number of all possible categories though, see e.g. [28, 1] for more examples of common categories.

Example 1 (The category of sets) In the literature, the archetypal first example of a category is the one describing the category of sets—a tradition this thesis obviously would be loath to break—and this is no coincidence. The definition is as simple as it is useful:

The category of sets, denoted `Set`, is defined such that its objects are sets and its morphisms are the functions between them. By using the regular notion of function composition and the identity function, the category conditions are fulfilled. □

Example 2 (The category of families of sets) In Definition 1, we gave a brief description of families of sets, or indexed sets. Similar to sets, these may also be extended to a category, say `FSet` which has families of sets as objects and families of functions as morphisms. Composition of morphisms is as previously described and the identity morphisms are trivial to construct. □

As the previous category demonstrated, the objects of a category do not necessarily have to be sets, they can be any mathematical structure, even other categories as will be shown later. The following example demonstrate another category which does not have sets as objects.

Example 3 (Viewing a partially ordered set as a category) A *partially ordered set*, commonly called a *poset*, is a pair $(S, \sqsubseteq)$, where S is a set and $\sqsubseteq$ is some binary relation over S which is

(i) reflexive, that is, $a \sqsubseteq a$;

(ii) transitive, that is, if $a \sqsubseteq b$ and $b \sqsubseteq c$, then $a \sqsubseteq c$; and

(iii) antisymmetric, that is, if $a \sqsubseteq b$ and $b \sqsubseteq a$, then $a = b$

for each $a, b, c \in S$.

This structure may be represented in a category `Pos` by letting $\mathrm{Ob}(\texttt{Pos}) = S$ and for each $a \sqsubseteq b$ have a morphism $a \xrightarrow{(a,b)} b$ in `Pos`. It is clear that this construction does indeed yield a category. The identity morphism is given by the reflexivity and the morphism composition is given by the transitivity property. Finally, it is easy to see that the composition is associative.

For a concrete example of this category, let us call it `Pos`$'$, consider the very simple poset with $S = \{1, 2, 4, 8\}$ and the regular $\leq$ relation over the natural numbers. Then `Pos`$'$ will consist of the objects 1, 2, 4, and 8 as well as the morphisms $1 \xrightarrow{(1,1)} 1$, $1 \xrightarrow{(1,2)} 2$, $1 \xrightarrow{(1,4)} 4$, $1 \xrightarrow{(1,8)} 8$, $2 \xrightarrow{(2,2)} 2$, $2 \xrightarrow{(2,4)} 4$, $2 \xrightarrow{(2,8)} 8$, $4 \xrightarrow{(4,4)} 4$, $4 \xrightarrow{(4,8)} 8$, and $8 \xrightarrow{(8,8)} 8$. Obviously, this is far from a convenient and manageable representation of the category. In an attempt to make the structure clearer, `Pos`$'$ can be represented in the following, more pleasant graphical form.

$$\begin{tikzcd}
1 \arrow[r, "(1,2)"] \arrow[rr, bend left=50, "(1,4)"] \arrow[rrr, bend left=70, "(1,8)"] \arrow[loop below, "(1,1)"] & 2 \arrow[r, "(2,4)"] \arrow[rr, bend left=50, "(2,8)"] \arrow[loop below, "(2,2)"] & 4 \arrow[r, "(4,8)"] \arrow[loop below, "(4,4)"] & 8 \arrow[loop below, "(8,8)"]
\end{tikzcd}$$

This figure may appear very much like a regular graph and, as will be shown in Section 2.1.1, this is no coincidence. □

It is possible to "lift" the previous example in the sense that while a poset may be represented as a category, it is also possible to represent the entire universe of posets in a category.

Example 4 (The category of partially ordered sets) Let `Poset` be the category of partially ordered sets. This category is constructed by letting its objects be all partially ordered sets and its morphisms contain the order-preserving (or monotonic) functions between them. For two posets, (S,α) and (T,β), an order preserving function, say $f : S \longrightarrow T$ is such that for $a,b \in S$, if $a\,\alpha\,b$, then $f(a)\,\beta\,f(b)$.

The regular identity function is clearly order-preserving and applying regular function composition to order-preserving functions yield an order-preserving function. We may therefore conclude that `Poset` indeed is a well-defined category. □

We will need a further notion when dealing with categories, namely the notion of opposite categories. Informally, this is is what one gets when taking a category and reversing all its morphisms.

Definition 3 (The dual of a category) The *opposite* (or *dual*) category of a category $\mathtt{C}$, denoted $\mathtt{C}^{op}$, is defined such that $\mathrm{Ob}(\mathtt{C}^{op}) = \mathrm{Ob}(\mathtt{C})$ and $\mathrm{Hom}_{\mathtt{C}}(A,B) = \mathrm{Hom}_{\mathtt{C}^{op}}(B,A)$, that is, for each $\mathtt{C}$-morphism $A \xrightarrow{f} B$, there is a $\mathtt{C}^{op}$-morphism $B \xrightarrow{f} A$.

As previously mentioned, this simply means that the original morphisms are reversed. When reversed, the identity morphisms in $\mathtt{C}$ will obviously yield the corresponding identity morphisms in $\mathtt{C}^{op}$. Morphism composition in $\mathtt{C}^{op}$, sometimes denoted $\circ^{op}$ is such that $g \circ f = f \circ^{op} g$. □

Example 5 In Example 3, it was shown that a category could represent a poset. The concrete example was the poset $(\{1,2,4,8\},\leq)$ which was represented by $\mathtt{Pos}'$. The natural question that arises is of course; what is $\mathtt{Pos}'^{op}$?

To illustrate the result we may simply copy the illustration shown in that example and reverse the morphisms. This yields the figure

$$\begin{tikzcd}
1 \arrow[loop below, "{(1,1)}"] & 2 \arrow[l, "{(1,2)}"'] \arrow[loop below, "{(2,2)}"] & 4 \arrow[l, "{(2,4)}"'] \arrow[ll, bend right=50, "{(1,4)}"'] \arrow[loop below, "{(4,4)}"] & 8 \arrow[l, "{(4,8)}"'] \arrow[ll, bend right=50, "{(2,8)}"'] \arrow[lll, bend right=70, "{(1,8)}"'] \arrow[loop below, "{(8,8)}"]
\end{tikzcd}$$

where it is easy to see that all morphisms go from a higher value to a lesser or equal one as opposed to greater or equal one which was the case in the original category. That is—disregarding the discrepancy in morphism labeling—$\mathtt{Pos}'^{op}$ represents the poset $(\{1,2,4,8\},\geq)$. □

Finally, it is possible to consider only a small section of a category. Informally, such a section is itself a category which consists of *some* objects from the full category and have as morphisms *some* of the morphisms that links these objects in the full category. The following definition formalize this idea:

Definition 4 (Subcategories) A *subcategory* to a category $\mathtt{C}$ is a category, $\mathtt{C}'$, such that $\mathrm{Ob}(\mathtt{C}') \subseteq \mathrm{Ob}(\mathtt{C})$ and $\mathrm{Hom}_{\mathtt{C}'}(A,B) \subseteq \mathrm{Hom}_{\mathtt{C}}(A,B)$ for each $A,B \in \mathrm{Ob}(\mathtt{C}')$. Additionally, we say that a subcategory is *full* if for all objects $A,B \in \mathrm{Ob}(\mathtt{C}')$, it is the case that $\mathrm{Hom}_{\mathtt{C}'}(A,B) = \mathrm{Hom}_{\mathtt{C}}(A,B)$. □

There are numerous other definitions and examples that may be presented and the ones given above only give a flavor of the breadth of mathematical structures that may be represented within this theoretical setting.

2.1.1 Diagrams

As was already hinted at in Example 3, it is possible to illustrate the structure of a category in the form of a graph. In this section this notion of representing a category in a graph will be formalized into a special type of graph which shall be called a diagram. The notion of commutativity—which is central to how diagrams and category theory relate—will also be defined.

Definition 5 (Diagrams) Given a category $\mathtt{C}$, let D be the directed graph such that

(i) for each vertex v in D there is an object D_v in $\mathtt{C}$ and

(ii) for each edge e with source v and target v' in D there is a morphism $D_v \xrightarrow{D_e} D_{v'}$ in $\mathtt{C}$,

then D will be referred to as the *diagram* of $\mathtt{C}$. □

This type of visualization is quite helpful and will be used a number of times from here on. To reduce clutter, the edges representing the identity morphisms will typically not be shown. It should should be remembered, however, that there always will be one for each object, it would not be a category if it were otherwise.

Of course, categories can grow quite large and are as can be imagined often unbounded in size—`Set` is an obvious example of this—and normally it is therefore the case that only a small section of the full diagram is shown. This is allowed by Definition 5 as it makes no requirement that all objects and morphisms of a category must be represented in the diagram.

So far, the definitions have only been used to produce representations of mathematical structures. It is now possible, using diagrams, to also introduce certain reasoning in a natural way. The notion of equations in the categorical setting is often illustrated with the help of diagrams. Consider, for example, the diagram

$$\begin{tikzcd}
A \arrow[d, "g"'] \arrow[dr, "f"] & \\
B \arrow[r, "h"'] & C ,
\end{tikzcd}$$

it can be seen that there are two possible paths from A to C, one being $A \xrightarrow{f} C$ and the other being the composition of $A \xrightarrow{g} B$ and $B \xrightarrow{h} C$, that is, $A \xrightarrow{h \circ g} C$. Since f and $h \circ g$ have the same domain and codomain, it is possible to formulate the equation $f = h \circ g$. This idea of letting diagrams represent equations is made formal by the definition that follows.

Definition 6 (Commutative diagrams) Let D be a diagram representing a category $\mathtt{C}$. Additionally, let $(f_1, f_2, \ldots, f_n)$ and $(g_1, g_2, \ldots, g_m)$ describe two paths in D between the nodes v and v' as the diagram

$$\begin{tikzcd}
 & u_1 \arrow[r, "f_2"] & u_2 \arrow[r, "\cdots"] & u_{n-1} \arrow[dr, "f_n"] & \\
v \arrow[ur, "f_1"] \arrow[dr, "g_1"'] & & & & v' \\
 & w_1 \arrow[r, "g_2"'] & w_2 \arrow[r, "\cdots"] & w_{m-1} \arrow[ur, "g_m"'] &
\end{tikzcd}$$

illustrates. Then D is said to be *commutative* if the corresponding compositions in $\mathtt{C}$ are equivalent, that is, if the equality

$$D_{f_n} \circ \ldots \circ D_{f_2} \circ D_{f_1} = D_{g_m} \circ \ldots \circ D_{g_2} \circ D_{g_1}$$

holds. □

As seen, diagrams can provide a good way to illustrate categories and equations in a categorical setting but not only that, they also give an easier way to reason about morphism composition. This is the case since a diagram is commutable if all its subdiagrams are likewise commutative [36]. As an example, suppose both the inner squares of the diagram

$$\begin{tikzcd}
A \arrow[r, "f"'] \arrow[rr, bend left, "g \circ f"] \arrow[d, "a"'] & B \arrow[r, "g"'] \arrow[d, "b"] & C \arrow[d, "c"] \\
A' \arrow[r, "f'"] \arrow[rr, bend right, "g' \circ f'"'] & B' \arrow[r, "g'"] & C'
\end{tikzcd}$$

commute, that is, $c \circ g = b \circ g'$ and $b \circ f = f' \circ a$. Then it is possible to directly draw the conclusion that also the outer square commutes and therefore $c \circ g \circ f = g' \circ f' \circ a$.

2.1.2 Morphisms

As we have seen, morphisms may be constructed in many different ways and it is often the case that the morphisms of one category have little in common with morphisms in another category. It is, however, possible to describe some general properties which are common to a wide variety of morphisms. These properties are in certain cases interesting enough that morphisms which have them are referred to by specific names.

In this section, a few morphisms like these are described. Beginning with the important *isomorphism* which may be recalled from the definition of regular functions.

Definition 7 (Properties of morphisms) A morphism $A \xrightarrow{f} B$ is

(i) an *isomorphism* if there exists a morphism $B \xrightarrow{g} A$ such that $g \circ f = \mathrm{id}_A$ and $f \circ g = \mathrm{id}_B$, we say that A and B are *isomorphic* and that g is the *inverse* of f;

(ii) an *endomorphism* if $A = B$, that is, the morphism domain and codomain are identical;

(iii) an *automorphism* if it both an endomorphism and isomorphism;

(iv) a *monomorphism* if for any two morphisms $C \underset{h}{\overset{g}{\rightrightarrows}} A$, it is the case that if $f \circ g = f \circ h$, then $g = h$.

(v) an *epimorphism* if for any two morphisms $B \underset{h}{\overset{g}{\rightrightarrows}} C$, it is the case that if $g \circ f = h \circ f$, then $g = h$. □

Example 6 In `Set`, the isomorphism are precisely the bijective functions while its mono- and epimorphisms are the surjective and injective functions, respectively. □

2.2 Functors

Just describing categories is of limited interest, the great power of category theory comes from the ability to describe transitions from one category to another. This type of transition is performed using constructions known as *functors*.

As an introductory example, we might consider `Poset`, the previously defined category of posets. Since a poset is a "set with structure", it is conceivable that a functor could be constructed which removes the structure from the posets and simply leaves us with the category `Set`. As Example 7 will show, the construction of this functor is quite simple. But, first it is necessary to formally define what a functor is.

Definition 8 (Functors) A *functor*—or more specifically a *covariant functor*—F from a category `C` to a category `D`, written $F : \mathtt{C} \longrightarrow \mathtt{D}$ or $\mathtt{C} \xrightarrow{F} \mathtt{D}$, is a collection of the functions

– $F_{\mathrm{Ob}} : \mathrm{Ob}(\mathtt{C}) \longrightarrow \mathrm{Ob}(\mathtt{D})$ and

– for each pair $A, B \in \mathrm{Ob}(\mathtt{C})$ the function

$$F_{A,B} : \mathrm{Hom}_{\mathtt{C}}(A,B) \longrightarrow \mathrm{Hom}_{\mathtt{D}}(F_{\mathrm{Ob}}(A), F_{\mathrm{Ob}}(B))$$

such that the following conditions hold:

(i) if $A \xrightarrow{f} B$ is in `C`, then $F(A \xrightarrow{f} B)$ is in `D` and

$$F(A \xrightarrow{f} B) = F_{\mathrm{Ob}}(A) \xrightarrow{F_{A,B}(f)} F_{\mathrm{Ob}}(B), \tag{2.1}$$

(ii) if $A \in \mathrm{Ob}(\mathtt{C})$, then

$$F_{A,A}(\mathrm{id}_A) = \mathrm{id}_{F_{\mathrm{Ob}}(A)}$$

(iii) if, for $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$, the morphism composition $g \circ f$ is in `C` then $F_{B,C}(g) \circ F_{A,B}(f)$ is in `D`. □

Typically, and in the rest of this text, the subscript is omitted, that is, F is used as notation for both F_{Ob} and $F_{A,B}$. Additionally the parenthesis are often disregarded—especially when the functor name consist of a single symbol. As a demonstration, these simplifications allow the right-hand side of Eq. (2.1) to be written

$$FA \xrightarrow{Ff} FB.$$

Now it is possible to properly construct the previously mentioned *forgetful functor*—named as such due to its forgetful nature where the output category contains less structural information that the input category—which takes `Poset` to `Set`. It should be mentioned that it is possible to construct this type of functor—one whose output category contains “less structure” than its input category—for many different categories and `Poset` is simply the example chosen for this text.

Example 7 (The forgetful functor taking `Poset` to `Set`) Let $F : \mathtt{Poset} \longrightarrow \mathtt{Set}$ be the forgetful function which shall be constructed. For each object $(S, \sqsubseteq) \in \mathrm{Ob}(\mathtt{Poset})$,

$$F_{\mathrm{Ob}}(S, \sqsubseteq) = S$$

and, for each morphism $f \in \mathrm{Hom}_{\mathtt{Poset}}(A, B)$

$$F_{A,B} f(x) = f(x)$$

for each $x \in A$. In other words, for each object in `Poset`, the F functor discards the order relation, leaving only the underlying set. $\square$

In the following chapters, the so-called *powerset functor* will be particularly important. Just to recall, the powerset of some set X is the set $\{x \mid x \subseteq X\}$, that is, the set of all subsets of X. This construction can be transferred to the world of functors as the following definition will show.

Example 8 (Powerset functor) The powerset functor $\mathcal{P}$ is a functor $\mathcal{P} : \mathtt{Set} \longrightarrow \mathtt{Set}$ defined such that for each set $A \in \mathrm{Ob}(\mathtt{Set})$, we have

$$\mathcal{P}A = \{x \mid x \subseteq A\},$$

and for each morphism $f \in \mathrm{Hom}(A, B)$, $A, B \in \mathrm{Ob}(\mathtt{Set})$, and $X \in \mathcal{P}A$, we have

$$\mathcal{P} f(X) = \{f(x) \in B \mid x \in X\}.$$

It might be of interest to remark that although this is the definition of the powerset functor which will be used in this text, there are other possible ways to construct a functor which may reasonably be called a powerset functor. They differ primarily in how the morphisms are handled [2]. $\square$

We may easily show two important results for functors; they are composable and this composition is associative. These two results are expressed in the following pair of propositions.

Proposition 1 *If $F : \mathtt{C} \longrightarrow \mathtt{D}$ and $G : \mathtt{D} \longrightarrow \mathtt{E}$ are functors then the composition $G \circ F : \mathtt{C} \longrightarrow \mathtt{E}$, often denoted by GF, is a functor*

$$(G \circ F)(A \xrightarrow{f} B) = G(FA) \xrightarrow{G(Ff)} G(FB).$$

$\square$

PROOF The result follow directly from the fact that functor composition is performed component-wise and as each component is a function, the normal function composition applies. That is,

$$\begin{aligned}(G \circ F)(A \xrightarrow{f} B) &= G(F(A \xrightarrow{f} B)) \\ &= G(FA \xrightarrow{Ff} FB) \\ &= G(FA) \xrightarrow{G(Ff)} G(FB)\end{aligned}$$

which establishes that the proposition is true. ■

Proposition 2 *Functor composition as given in Proposition 1 is associative.* □

PROOF Let $X \xrightarrow{f} Y$ be a morphism in $\mathtt{A}$ and let $\mathtt{A} \xrightarrow{F} \mathtt{B}$, $\mathtt{C} \xrightarrow{G} \mathtt{D}$ as well as $\mathtt{D} \xrightarrow{H} \mathtt{E}$ be functors. We wish to show that

$$(H \circ (G \circ F))(X \xrightarrow{f} Y) = ((H \circ G) \circ F)(X \xrightarrow{f} Y).$$

Again using the fact that functor composition is performed component-wise and each component is a function, the result follows directly from the associativity of function composition. That is,

$$\begin{aligned}(H \circ (G \circ F))(X \xrightarrow{f} Y) &= (H \circ (G \circ F))X \xrightarrow{(H\circ(G\circ F))f} (H \circ (G \circ F))Y \\ &= ((H \circ G) \circ F)X \xrightarrow{((H\circ G)\circ F)f} ((H \circ G) \circ F)Y \\ &= ((H \circ G) \circ F)(X \xrightarrow{f} Y)\end{aligned}$$

and we have by this shown that the proposition holds. ■

Note that in Proposition 1 and 2, we have used the notational simplification of functors—that is, both F_{Ob} and $F_{A,B}$ are written F—as the claims would otherwise become rather unreadable, in particular the derivations in the proofs. The diligent reader may find it an interesting exercise to redo the derivations with subscripts in place.

The definition of functors combined with above results regarding functor composition allow us to create the *category of categories*, `Cat`, in which the objects are all categories[2] and the morphisms are the functors between them. The identity functor—defined in short by

$$\mathrm{id}_{\mathtt{C}}(A \xrightarrow{f} B) = A \xrightarrow{f} B$$

for each category $\mathtt{C}$ where $A, B \in \mathrm{Ob}(\mathtt{C})$ and $f \in \mathrm{Hom}_{\mathtt{C}}(A, B)$—act as identity morphism in `Cat`.

In Definition 8 it was said that a functor technically is called a *covariant functor*. In the following definition, the dual of covariant functor is presented; the *contravariant functor*. The contravariant functor is less commonly used so when it is referred to, the full name is always given as opposed to the “regular”, covariant, functor which often misplace its first name.

[2]Strictly speaking, the categories of `Cat`, and `Cat` itself, are *small categories*. Only small categories are considered in this thesis so the distinction between small and large will be mysteriously glossed over.

Definition 9 (Contravariant functors) A *contravariant functor* $F : \mathtt{C} \longrightarrow \mathtt{D}$ is defined in much the same way as its covariant namesake with the exception that for each morphism $f : A \longrightarrow B$ in $\mathtt{C}$, the morphism

$$F(A \xrightarrow{f} B) = FB \xrightarrow{Ff} FA$$

is in $\mathtt{D}$. That is, the morphism is reversed. This reversion may be equivalently expressed using the covariant functor $G : \mathtt{C}^{op} \longrightarrow \mathtt{D}$ or $G : \mathtt{C} \longrightarrow \mathtt{D}^{op}$. Similarly, given a covariant functor $H : \mathtt{C} \longrightarrow \mathtt{D}$, H^{op} denotes the corresponding contravariant functor. That is, F, G and H^{op} may represent the same contravariant functor! To avoid confusion, only the latter two notations will be used in this text. □

As a concrete example, one might consider the contravariant version of the powerset functor or as was mentioned in Definition 9, the following equivalent covariant functor.

Example 9 Let $Q : \mathtt{Set} \longrightarrow \mathtt{Set}^{op}$ be a functor such that for each combination of sets $A, B \in \mathrm{Ob}(\mathtt{Set})$, morphism $A \xrightarrow{f} B$ and $Y \in \mathcal{P}B$. We then have

$$QA = \mathcal{P}A,\ QB = \mathcal{P}B, \text{ and } Qf(Y) = \{x \in A \mid f(x) \in Y\}.$$

In other words, Qf when applied to the set Y yields the union of each preimage of elements in Y. □

2.3 Natural Transformations

The final major area of basic category theory which shall be presented is the notion of *natural transformations*. These are to functors as functors are to categories. By this is meant that a natural transformation takes a functor as input and yield another functor as output and as we shall see, it is possible to construct a category of functors in which natural transformations constitute the morphisms.

As an aside, for those curious about the monads mentioned in the chapter introduction, very informally they may be explained as a triple containing one functor and two natural transformations such that a few conditions are upheld. While not used in this text, it is—since they have shown themselves so valuable in various applications—well worth knowing that they arose from the field of category theory.

Definition 10 (Natural transformations) For two functors $\begin{tikzcd}\mathtt{C} \arrow[r, shift left, "F"] \arrow[r, shift right, "G"'] & \mathtt{D}\end{tikzcd}$, a *natural transformation* τ from F to G, denoted by the now familiar $\tau : F \longrightarrow G$ or $F \xrightarrow{\tau} G$, is a family of $\mathtt{D}$-morphisms $\tau = (\tau_A)_{A \in \mathrm{Ob}(\mathtt{C})}$ where $\tau_A : FA \longrightarrow GA$ and the diagram

$$\begin{tikzcd}
A \arrow[d, "f"'] & FA \arrow[r, "\tau_A"] \arrow[d, "Ff"'] & GA \arrow[d, "Gf"] \\
B & FB \arrow[r, "\tau_B"'] & GB
\end{tikzcd}$$

commutes. That is, $Gf \circ \tau_A = \tau_B \circ Ff$ for each $\mathtt{C}$-morphism $A \xrightarrow{f} B$ with $A, B \in \mathrm{Ob}(\mathtt{C})$. This is known as the *naturality condition on* τ [2]. □

Remark 4 We may for any functor F construct its *identity natural transformation*, $F \xrightarrow{\mathrm{id}_F} F$, defined to be $(\mathrm{id}_F)_A = \mathrm{id}_{FA}$. □

Example 10 Consider the powerset functor, $\mathtt{Set} \xrightarrow{\mathcal{P}} \mathtt{Set}$, as given in Example 8. Is there a reasonable way to construct a natural transformation $\mathrm{id}_{\mathtt{Set}} \xrightarrow{\eta} \mathcal{P}$? That is, we wish to find an η such that we, for each $A \in \mathrm{Ob}(\mathtt{Set})$, have $A \xrightarrow{\eta_A} \mathcal{P}A$. Let us try with $\eta_A(x) = \{x\}$ for each $x \in A$. It remains to show that this definition really make η fulfill the naturality condition, that is, does

$$\begin{tikzcd} A \arrow[d, "f"'] & A \arrow[r, "\eta_A"] \arrow[d, "f"'] & \mathcal{P}A \arrow[d, "\mathcal{P}f"] \\ B & B \arrow[r, "\eta_B"'] & \mathcal{P}B \end{tikzcd}$$

commute? A simple derivation shows that it does; let $x \in A$, then

$$(\mathcal{P}f \circ \eta_A)(x) = \mathcal{P}f(\eta_A(x)) = \mathcal{P}f(\{x\}) = \{f(x)\} = \eta_B(f(x)) = (\eta_B \circ f)(x).$$

□

Example 11 In a manner similar to the previous example, we may easily construct a natural transformation $\mathcal{P}\mathcal{P} \xrightarrow{\mu} \mathcal{P}$ by letting $\mu_A(X) = \bigcup_{x \in X} x$ for each $X \in \mathcal{P}\mathcal{P}A$. Showing that μ is a natural transformation is again very similar to the previous example.

As the examples show, natural transformations are conceptually not very difficult even though it may seem so looking at Definition 10. Composition is not difficult either, since a natural transformation is a family of morphisms it is not surprising that composition of natural transformations is performed component-wise. This type of composition goes by the name of *vertical composition*. As this name suggests—and as will be seen shortly—there is also a horizontal form of composition.

Definition 11 (Vertical composition) For the natural transformations $F \xrightarrow{\tau} G$ and $G \xrightarrow{\tau'} H$ there is a composition $\tau' \circ \tau$ such that $F \xrightarrow{\tau' \circ \tau} H$. Known as *vertical composition*, it is defined

$$(\tau' \circ \tau)_A = \tau'_A \circ \tau_A.$$

In the literature, $\tau'\tau$ is a common alternative to the $\tau' \circ \tau$ notation. □

As a natural transformation is a family of morphisms, associativity of vertical composition is immediate. Before tackling horizontal composition, we first need to consider how functors and natural transformations may be combined. The following definition establishes how these combinations are constructed.

Definition 12 For the functors $\mathtt{C} \underset{G'}{\overset{G}{\rightrightarrows}} \mathtt{D}$, $\mathtt{E} \xrightarrow{F} \mathtt{C}$, and $\mathtt{D} \xrightarrow{F'} \mathtt{E}$ as well as the natural transformation $G \xrightarrow{\tau} G'$ it is the case that

– $GF \xrightarrow{\tau F} G'F$ is defined to be $(\tau F)_A = \tau_{FA}$;

– $F'G \xrightarrow{F'\tau} F'G$ is defined to be $(F'\tau)_A = F'\tau_A$; and

both τF and $F'\tau$ are natural transformations. □

Definition 13 (Horizontal composition) For two natural transformations $F \xrightarrow{\tau} G$ and $F' \xrightarrow{\tau'} G'$ there is a *horizontal composition* (or *star composition*) $\tau' \star \tau$ such that $F'F \xrightarrow{\tau' \star \tau} G'G$. The horizontal composition can be defined using the commutativity of the diagram

$$\begin{tikzcd}
 & G'FX \arrow[dr, "G'\tau_X"] & \\
F'FX \arrow[ur, "\tau'_{FX}"] \arrow[dr, "F'\tau_X"'] \arrow[rr, dashed, "\tau' \star \tau"] & & G'GX \\
 & F'GX \arrow[ur, "\tau'_{GX}"'] &
\end{tikzcd} .$$

That is, the horizontal composition is defined as

$$\tau' \star \tau = \tau' G \circ F'\tau = G'\tau \circ \tau' F. \tag{2.2}$$

□

Remark 5 It is easy to show that the combination of functors and natural transformations as shown in Definition 12 may be represented as horizontal combination using identity natural transformations. Let F be a functor and τ a natural transformation. Then $F\tau = \mathrm{id}_F \star \tau$ and $\tau F = \tau \star \mathrm{id}_F$. □

Using these notions of natural transformations it is now a simple matter of concluding that it is possible to create a category with functors as objects and natural transformations as morphisms.

Definition 14 (The category of functors) Let $\mathtt{C}$ and $\mathtt{D}$ be categories, then we can form a category $\mathtt{Fun}(\mathtt{C}, \mathtt{D})$ with functors from $\mathtt{C}$ to $\mathtt{D}$ as objects and the natural transformations between them as morphisms. The associativity condition is upheld as per the earlier remark. The identity morphisms are precisely the identity natural transformations. □

This concludes our introduction to category theory, we have considered categories themselves and the functors which operate on them and lastly the natural transformations which operate on functors. Although only a minute part of the field of category theory has been presented, these concepts together form the foundations of the category theory needed for the understanding of this thesis. We have, in fact, covered a little bit more than what is strictly needed.

As we have seen, a category is a very general concept and as a result, it is perhaps not immediately obvious why it relevant to the topic of this thesis or even why it is relevant at all. The following chapters will hopefully provide some clarification as to why this categorical approach is a sound one and furthermore show that it allows an elegant and natural way of explaining the rather intricate concepts involved.

Chapter 3

Universal Algebra

Being able to accurately apply a specific reasoning in a great number of situations is an important research goal in many areas of mathematics and computing science. By having the ability to prove that a certain line of thought is true for all cases—even previously unknown ones—allows much work to be avoided and subsequently efforts can be placed in areas where they are more needed. A well known theorem in the area of computability theory may be taken as an example, namely Rice's theorem [38]. This theorem in essence states that the problem of deciding whether or not a program has some non-trivial property is undecidable in the general case—a property, such as whether a program will compute a specific function, is trivial if either all programs have the property or no program has it. This is a very strong and useful claim since it applies to all programs and all non-trivial properties. In this chapter we will examine how to create a situation where it is possible to do similar broad reasoning concerning abstract algebra and logic.

Briefly stated, abstract algebra is the mathematical field studying algebraic structures such as the previously mentioned groups but also vector spaces for example—linear algebra may subsequently be considered a branch of abstract algebra. Henceforth, for simplicity and in line with most of the current literature, abstract algebra will often be referred to simply by *algebra*. Care should therefore be taken to not confuse algebra in the meaning of abstract algebra and algebra in the meaning of elementary algebra which concerns manipulation of numeric formulas containing unknown values.

The particular field of abstract algebra which goes under the name *universal algebra* is the topic of this chapter. Universal algebra allows us to describe and reason about the syntax and semantics of algebraic expressions in a general manner. The advantage of being able to describe these expressions is that it allows us to construct new "languages of mathematics" which may be used in a uniform manner, that is we are not required to invent a new formalism each time we create a new algebra. As will become apparent, this is very useful when one, for example, wish to define a new algebra in terms of an already existing algebra.

We will in this chapter briefly look at both the syntactic aspects and the semantic aspects of universal algebra. While the former refer to the appearance of the algebraic expressions we wish to use—in essence, how to write them down on paper—the latter is about the meaning we place into this appearance. As a result of separating these two notions it is, as we shall see, easily possible to place drastically different meaning into the expressions we write. Both the syntactic and the semantic notions are certainly of great importance when constructing an algebra and the way in which we describe them will therefore be investigated in some detail.

The type of universal algebra considered in this section is of a *many-sorted* variant which in short means that operations in the algebra limit themselves to certain *sorts*. As a familiar example, integer addition may be considered: The integer addition operation is defined such that it takes two integers as input and give a third integer as output. It would therefore not make sense to, for example, provide an integer and a boolean value as input to this operation. As will be seen, these notions will emerge quite naturally within the framework of many-sorted algebras described in this section.

The seasoned reader may notice that the notions presented here does not make full use of *algebraic theories* [29]. Instead, the content in this chapter is primarily based upon the more straightforward notions found in [27, 4, 17] but adapted to the categorical vocabulary where applicable.

3.1 Signatures

In order to write down algebraic expressions we need some rules as to what constitute a "valid" expression within this algebra. This is analogous to the familiar situation of programming languages in which there is a—more or less—formally defined syntax. This syntax determine which strings are valid instances of programs in that language. In the case of algebras the formal syntax is described using a simple mathematical structure known as a *signature* which will be defined presently and is followed by a number of examples.

Definition 15 (Signatures) A signature, Σ, is a pair $\Sigma = (S, \Omega)$ where

- S is a set of *sorts* (or *types*), and
- Ω is a set of *operations* where an n-ary operation having *operation name* ω is an $(n+2)$-tuple $(\omega, s_1, \ldots, s_n, s)$ with $n \in \mathbb{N}$ and $s, s_1, \ldots, s_n \in S$. Rather than using the—in this context—clumsy tuple notation we shall denote the operation by the more clear

$$\omega : s_1 \times \ldots \times s_n \longrightarrow s$$

instead, or even just ω if the context allows.

We say that a signature is *strongly typed* if for any two identically named operations in a signature, say $\omega : s_1 \times \ldots \times s_n \longrightarrow s$ and $\omega : s'_1 \times \ldots \times s'_k \longrightarrow s'$, it is the case that if $n = k$ then there exists at least one i such that $s_i \neq s'_i$. All signatures used in this text are assumed to be strongly typed. □

Remark 6 In the following chapters, the notion of a signature will be generalized somewhat and this type of signature will be referred to as an *equational signature*. However, throughout this chapter only equational signatures will be considered and they will therefore, in the name of simplicity, here be referred to simply as signatures. □

Remark 7 Please note that, at this point, there is no semantic meaning put into the

$$\omega : s_1 \times \ldots \times s_n \longrightarrow s$$

notation, that is, while it may appear so, it does not constitute a function definition or anything similar; it is merely a syntactic object. The case of a 0-ary (or *nullary*) operation $\omega : \longrightarrow s$ is therefore perfectly valid and, as we will see later, represents a single element of sort s. In other words; a *constant*. □

Often, we wish to refer to the operation names of a signature and as a convenient notation for this we introduce the convention that $\mathrm{Op}(\Sigma)$ stands for the set of operation names of a signature $\Sigma = (S, \Omega)$. More formally we may state that

$$\mathrm{Op}(\Sigma) = \{\mathrm{proj}_1(\omega) \mid \omega \in \Omega\}$$

where proj_1 projects the first component of an operation, that is, the operation name. Another convenient notation is to, given an operation

$$\omega : \underbrace{s \times s \times \ldots \times s}_{\text{n times}} \longrightarrow s'$$

write $\omega : s^n \longrightarrow s'$.

Using Definition 15, it is possible to easily construct useful signatures as illustrated by the following examples. First, we create the signature of natural numbers.

Example 12 (Signature of natural numbers) One of the least complicated but still extremely fundamental signature is the one describing the natural numbers. This signature, denoted $\Sigma_{\mathrm{NAT}} = (S_{\mathrm{NAT}}, \Omega_{\mathrm{NAT}})$, is constructed by letting

$$\begin{aligned} S_{\mathrm{NAT}} &= \{nat\}, \\ \Omega_{\mathrm{NAT}} &= \{0 : \longrightarrow nat, \\ &\qquad Succ : nat \longrightarrow nat\}. \end{aligned}$$

Please note that 0 is a nullary symbol—a constant—and as such it does therefore not accept any argument. The set of operation names is in this case $\mathrm{Op}(\Sigma_{\mathrm{NAT}}) = \{0, Succ\}$. As will be seen in Example 14, we could also introduce constants 1, 2, and so forth.

$\square$

In a similar fashion, it is easy to create the signature of booleans:

Example 13 (Signature of booleans) The signature of booleans, which will be denoted $\Sigma_{\mathrm{BOOL}} = (S_{\mathrm{BOOL}}, \Omega_{\mathrm{BOOL}})$, is familiar to many and a simple definition may be such that

$$\begin{aligned} S_{\mathrm{BOOL}} &= \{bool\}, \\ \Omega_{\mathrm{BOOL}} &= \{True : \longrightarrow bool \\ &\qquad False : \longrightarrow bool \\ &\qquad \neg : bool \longrightarrow bool \\ &\qquad \wedge : bool \times bool \longrightarrow bool\}. \end{aligned}$$

$\square$

We may combine two signatures by simply performing a component-wise union. We may also add further sorts and operations by simply adding them using another union operation. The following example defines a signature which demonstrates this. Note that this signature in particular will be used extensively in examples throughout this text.

Example 14 (The NATBOOL **signature)** This signature is constructed by combining the above signatures Σ_{NAT} and Σ_{BOOL} and adding the binary operator $\leq$ which makes use of the fact that both *bool* and *nat* is available in this new signature. Finally, the binary operations $+$ and $*$ are added together with all non-zero natural number constants.

$$\begin{aligned}
S_{\text{NATBOOL}} &= S_{\text{NAT}} \cup S_{\text{BOOL}}, \\
\Omega_{\text{NATBOOL}} &= \{\ \leq : nat \times nat \longrightarrow bool \\
&\qquad + : nat \times nat \longrightarrow nat \\
&\qquad * : nat \times nat \longrightarrow nat \\
&\qquad 1 : \longrightarrow nat \\
&\qquad 2 : \longrightarrow nat \\
&\qquad \vdots \\
&\quad \} \cup \Omega_{\text{NAT}} \cup \Omega_{\text{BOOL}}.
\end{aligned}$$

□

A signature may be viewed as the definition of an abstract data type—where, as we shall see, an algebra will act as the implementation of the data type. The previous examples act as illustrations of this but the following example will highlight it in a more concrete manner. In this example, we extend the Σ_{NATBOOL} signature in such a way that also lists of natural numbers may be expressed.

Example 15 (List of natural numbers) Let $\Sigma_{\text{NATLIST}} = (S_{\text{NATLIST}}, \Omega_{\text{NATLIST}})$ be the aforementioned signature for the list of natural numbers. Then one possible definition of this abstract data type may be

$$\begin{aligned}
S_{\text{NATLIST}} &= \{natlist\} \cup S_{\text{NATBOOL}}, \\
\Omega_{\text{NATLIST}} &= \{Nil : \longrightarrow natlist \\
&\qquad Head : natlist \longrightarrow nat \\
&\qquad Tail : natlist \longrightarrow natlist \\
&\qquad Cons : nat \times natlist \longrightarrow natlist \\
&\qquad Empty : natlist \longrightarrow bool\} \cup \Omega_{\text{NATBOOL}}.
\end{aligned}$$

□

A very useful—it is, in fact, quite essential—extension to the notion of signatures as given in Definition 15 is the addition of *variables*. These allow algebraic expressions to be extensively generalized. They also allow the introduction of unification which traditionally has been of supreme importance within computing science.[1]

Definition 16 (Signature with variables) Let $\Sigma = (S, \Omega)$ be a signature, then there exists an associated family of infinite sets $\chi = (\chi_s)_{s \in S}$ where each $x \in \chi_s$ is called a *variable of sort s* and an $X \subseteq \chi$ is said to be a *family of variables* for the signature Σ.

We restrict the construction of X by requiring that

(i) it is pairwise disjoint so as to avoid the situation where a variable may be given multiple sorts; and

(ii) no variable conflicts with any operation symbol, that is,

$$\text{Op}(\Sigma) \cap \bigcup_{s \in S} \chi_s = \varnothing.$$

□

[1]This text will only peripherally touch the subject of unification. For a categorical treatment of the area please see e.g. [14, 39].

Remark 8 Since χ in the unsorted case is a set and by extension so is $X \subseteq \chi$, it has traditionally been the case that X has been called a *set of variables* even when associated with a many-sorted signature. While—stemming from the restriction that X is pairwise disjoint—there is some rationality to the "set of variables" phrase even in the many-sorted case, this tradition will be forgone in this thesis for the purpose of stressing the point that we are working with many-sorted signatures. $\square$

Remark 9 From now on, it is assumed that all signatures are extended to include variables as they are presented in the previous definition. $\square$

Just building signatures is not very fruitful. It is of interest to describe transitions from one signature to another and examining the properties of these transitions. Eventually, this will allow the construction of a category of signatures which will prove to be quite useful later on. This quest will begin by defining the notion of a signature morphism, which in short takes a given signature and produces another one with the operation arities intact.

Definition 17 (Signature morphisms) If $\Sigma = (S, \Omega)$ and $\Sigma' = (S', \Omega')$ are signatures, then a signature morphism $\mu : \Sigma \longrightarrow \Sigma'$, or $\Sigma \xrightarrow{\mu} \Sigma'$ is a pair of functions

$$\begin{aligned} \mu = (&\mu_S : S \longrightarrow S', \\ &\mu_\Omega : \Omega \longrightarrow \Omega') \end{aligned}$$

such that for each $\omega : s_1 \times \ldots \times s_n \longrightarrow s \in \Omega$,

$$\mu_\Omega(\omega : s_1 \times \ldots \times s_n \longrightarrow s) = \omega' : \mu_S(s_1) \times \ldots \times \mu_S(s_n) \longrightarrow \mu_S(s)$$

for some $\omega' : \mu_S(s_1) \times \ldots \times \mu_S(s_n) \longrightarrow \mu_S(s) \in \Omega'$.

Additionally, if X is a family of variables associated to Σ and X' is a family of variables similarly associated to Σ', then we associate to the signature morphism an additional family of injective functions $\mu_X : X \longrightarrow X'$. These functions are required to be injective in order to guarantee that a unique Σ-variable is taken to a unique Σ'-variable.

For simplicity, each of μ_S, μ_Ω, and μ_X will be denoted μ where context allows and the risk of confusion is minimal. $\square$

Example 16 One of the most trivial but still useful signature morphisms possible is—as is often the case—the identity signature morphism, that is, $id_\Sigma : \Sigma \longrightarrow \Sigma$ which is trivially constructed by letting

$$\begin{aligned} id_S(s) &= s, \text{ and} \\ id_\Omega(\omega) &= \omega \end{aligned}$$

for each $s \in S$ and $\omega \in \mathrm{Op}(\Sigma)$. $\square$

Another simple type of signature morphism is an inclusion signature morphism, say $\iota : \Sigma \longrightarrow \Sigma'$ taking a signature $\Sigma = (S, \Omega)$ to the "larger" signature $\Sigma' = (S', \Omega')$. Larger here simply means that $S \subseteq S'$ and $\Omega \subseteq \Omega'$. It is then the case that ι_S and ι_Ω are defined much like id_S and id_Ω above—the only difference being their codomain.

A slightly more complex situation is when the signature morphism is required to perform *renaming* as in the following example.

Example 17 Let $\Sigma_{\mathrm{NATBOOL}}$ be as given in Example 14 and let $\Sigma_{\mathrm{NAT}'} = (S_{\mathrm{NAT}'}, \Omega_{\mathrm{NAT}'})$ be such that

$$\begin{aligned} S_{\mathrm{NAT}'} &= \{natural\} \\ \Omega_{\mathrm{NAT}'} &= \{Zero : \longrightarrow natural \\ &\quad\ Succ : natural \longrightarrow natural \\ &\quad\ Add : natural \times natural \longrightarrow natural\}. \end{aligned}$$

We may then construct a signature morphism $\mu : \Sigma_{\mathrm{NAT}'} \longrightarrow \Sigma_{\mathrm{NATBOOL}}$ by letting

$$\begin{aligned} \mu_S(natural) &= nat, \\ \mu_\Omega(Zero) &= 0, \\ \mu_\Omega(Succ) &= Succ, \text{ and} \\ \mu_\Omega(Add) &= +. \end{aligned}$$

Here, the condensed form of writing operations have been used. Writing, for example, the final application of μ_Ω in full gives

$$\begin{aligned} &\mu_\Omega(Add : natural \times natural \longrightarrow natural) \\ &\quad = + : \mu_S(natural) \times \mu_S(natural) \longrightarrow \mu_S(natural) \\ &\quad = + : nat \times nat \longrightarrow nat \end{aligned}$$

which clearly shows that $\mu_\Omega(Add : natural \times natural \longrightarrow natural) \in \Omega_{\mathrm{NATBOOL}}$. Note that neither μ_S or μ_Ω are surjective in this case and that this is no requirement which has been placed on signature morphisms. □

Definition 18 (Composition of signature morphisms) For two signature morphisms $\mu : \Sigma_1 \longrightarrow \Sigma_2$ and $\nu : \Sigma_2 \longrightarrow \Sigma_3$, the composed morphism $\nu \circ \mu : \Sigma_1 \longrightarrow \Sigma_3$ is defined such that

$$\begin{aligned} (\nu \circ \mu)(s) &= \nu(\mu(s)) \text{ and} \\ (\nu \circ \mu)(\omega : s_1 \times \ldots \times s_n \longrightarrow s) &= \omega'' : (\nu \circ \mu)(s_1) \times \ldots \times (\nu \circ \mu)(s_n) \longrightarrow (\nu \circ \mu)(s) \end{aligned}$$

for each $s \in S_{\Sigma_1}$ and $\omega \in \Omega_{\Sigma_1}$ and where ω'' is such that $\nu(\mu(\omega)) = \omega''$. In other words, composition is simply performed component-wise. □

The fact that composition of signature morphisms is defined in terms of regular functions leads to the following proposition in a natural way.

Proposition 3 (Composition of signature morphisms is associative) *Given the signature morphisms $\mu : \Sigma_1 \longrightarrow \Sigma_2$, $\nu : \Sigma_2 \longrightarrow \Sigma_3$, and $\xi : \Sigma_3 \longrightarrow \Sigma_4$, it is the case that*

$$\xi \circ (\nu \circ \mu) = (\xi \circ \nu) \circ \mu,$$

that is, composition of signature morphisms is associative. □

PROOF Let $\omega : s_1 \times \ldots \times s_n \longrightarrow s$ be an arbitrary operation in Σ_1. Then using associativity of regular function composition and Definition 17 it is trivially the case that

$$\begin{aligned} &(\xi \circ (\nu \circ \mu))(\omega : s_1 \times \ldots \times s_n \longrightarrow s) \\ &\quad = (\xi \circ (\nu \circ \mu))(\omega) : (\xi \circ (\nu \circ \mu))(s_1) \times \ldots \times (\xi \circ (\nu \circ \mu))(s_n) \longrightarrow (\xi \circ (\nu \circ \mu))(s) \\ &\quad = ((\xi \circ \nu) \circ \mu)(\omega) : ((\xi \circ \nu) \circ \mu)(s_1) \times \ldots \times ((\xi \circ \nu) \circ \mu)(s_n) \longrightarrow ((\xi \circ \nu) \circ \mu)(s) \\ &\quad = ((\xi \circ \nu) \circ \mu)(\omega : s_1 \times \ldots \times s_n \longrightarrow s). \end{aligned}$$

■

We are now, using Definition 15, Remark 16, and Proposition 3 able to conclude that there is a category of signatures.

Definition 19 (The category of signatures) We define the category of (equational) signatures, $\texttt{Sign}^{Eq}$, by letting

- $\mathrm{Ob}(\texttt{Sign}^{Eq})$ be the set of all signatures as presented in Definition 15, and
- for each signature $A, B \in \mathrm{Ob}(\texttt{Sign}^{Eq})$, $\mathrm{Hom}_{\texttt{Sign}^{Eq}}(A, B)$ is the set of all signature morphisms from A to B. □

3.2 Terms

From a signature, it is possible to construct *terms* which are, in the programming language analogy, similar to expressions using the abstract data types defined using the signature. Using terms it is possible to combine operations freely under the condition that the associated sorts are respected and this combination is done in a way that appear very much like mathematical function application. The practical appearance of this will be made clear after the following, more formal, definition.

Definition 20 (Terms) Let $\Sigma = (S, \Omega)$ be a signature with X being an associated family of variables, then there is a family of sets $T_\Sigma X = (T_{\Sigma,s}X)_{s\in S}$ whose members are called *terms* and which constitute strings over the alphabet

$$\bigcup_{s\in S} X_s \cup \mathrm{Op}(\Sigma) \cup \{(,)\} \cup \{,\}.$$

More specifically, a term $t \in T_{\Sigma,s}X$ for some $s \in S$ is said to be a *term of sort s*. For each $s \in S$, the set $T_{\Sigma,s}X$ is inductively defined such that

(i) for each variable $x \in X_s$, we have $x \in T_{\Sigma,s}X$;

(ii) for each constant $\omega : \longrightarrow s \in \Omega$, we have $\omega \in T_{\Sigma,s}X$; and

(iii) for each non-nullary operation $\omega : s_1 \times \ldots \times s_n \longrightarrow s \in \Omega$, it is the case that $\omega(t_1, \ldots, t_n) \in T_{\Sigma,s}X$ provided $n \geq 1$, $s_i \in S$, and $t_i \in T_{\Sigma,s_i}X$ for $1 \leq i \leq n$.

As a special case, if $X = (\varnothing_s)_{s\in S}$, then $T_\Sigma X$ is the family of sets containing only variable-free terms—also known as *ground terms*.

We will, for some term t, have $\mathrm{var}(t)$ denote the family of variables which occur in that term. □

Example 18 Let the signature Σ_{BOOL} be as given in Example 13. Then, from Definition 20(ii), we have the case that $True$ and $False$ are terms. From Definition 20(iii) is is clear that $\neg(True)$ and $\neg(False)$ are terms, as are $\wedge(True, True)$, $\wedge(False, \neg(True))$, and so forth.

If we further let $X = \{\{x, y, z\}_{bool}\}$ be variables of sort $bool$. Then from Definition 20(i), it is known that x, y and z are terms as well. We may of course mix all these as we please—again, provided the operation sorts are respected—and thus the string

$$\wedge(\wedge(x, False), \neg(y)) \tag{3.1}$$

is yet another term and $\mathrm{var}[\wedge(\wedge(x, False), \neg(y))] = \{\{x, y\}_{bool}\}$. Each of these example terms are of course of the sort $bool$. □

To make terms more familiar, we may introduce some syntactic sugar in the form of infix expressions. That is, the term $\wedge(\wedge(x, False), \neg(y))$ may be written in the equivalent form $(x \wedge False) \wedge (\neg y)$ or even $x \wedge False \wedge \neg y$. A formal description of this notation will be omitted but in short the operators, including operator priorities, will work in the commonly understood way.

The notion of signature morphisms may be extended to terms and will then form so-called *term morphisms*. Intuitively, if there exists a signature morphism taking one signature to another, then we may form a term morphism which take terms of the first signature into terms of the second signature. In other words, the translation occurs in the same direction as the signature morphism.

Definition 21 (Term morphisms) Let $\Sigma = (S, \Omega)$ and $\Sigma' = (S', \Omega')$ be two signatures, $\mu : \Sigma \longrightarrow \Sigma'$ a signature morphism between them, and finally let X be a family of variables for Σ and

$$X' = \{(\mathcal{P}\mu(X_s))_{\mu(s)} \mid s \in S\}$$

be a family of variables for Σ'. The term morphism, $\mu_T : T_\Sigma X \longrightarrow T_{\Sigma'} X'$, taking a Σ-term to a Σ'-term, is then inductively defined such that

(i) if $t = x$ for some $x \in X$, then $\mu_T(t) = \mu_X(x)$;

(ii) if $t = \omega$ for some $\omega : \longrightarrow s \in \Omega$ and

$$\mu_\Omega(\omega : \longrightarrow s) = \omega' : \longrightarrow \mu_S(s),$$

then $\mu_T(t) = \omega'$; and

(iii) if $t = \omega(t_1, \ldots, t_n)$ for some $n \geq 1$, $\omega : s_1 \times \ldots \times s_n \longrightarrow s \in \Omega$, and $t_i \in T_{\Sigma, s_i} X$ with $1 \leq i \leq n$ and where

$$\mu_\Omega(\omega : s_1 \times \ldots \times s_n \longrightarrow s) = \omega' : \mu_S(s_1) \times \ldots \times \mu_S(s_n) \longrightarrow \mu_S(s),$$

then $\mu_T(t) = \omega'(\mu_T(t_1), \ldots, \mu_T(t_n))$.

For simplicity, we denote μ_T by μ when context allows. □

From this definition, it is not immediately obvious that the term morphism gives a correct term, that is, that given the conditions in Definition 21 and $t \in T_\Sigma X$ it must be the case that $\mu(t) \in T_{\Sigma'} X'$. Fortunately, the correctness of this morphism may easily be established as the following proposition shows.

Proposition 4 *Let $\Sigma = (S, \Omega)$ and $\Sigma' = (S', \Omega')$ be two signatures and X be a family of Σ-variables. Further, let $X' = \{(\mathcal{P}\mu(X_s))_{\mu(s)} \mid s \in S\}$ be a family of Σ' variables, $\mu : \Sigma \longrightarrow \Sigma'$ be a signature morphism and $\mu_T : T_\Sigma X \longrightarrow T_{\Sigma'} X'$ be a term morphism constructed as specified in Definition 21. Then*

$$t \in T_\Sigma X \iff \mu_T(t) \in T_{\Sigma'} X'.$$ □

PROOF The proof is performed by straight-forward induction. Let $t \in T_\Sigma X$ be some Σ-term, then

– if $t = x$ for some variable $x \in X$, then by Definition 21(i) we have $\mu_T(t) = \mu_X(x) \in X'$ which by Definition 20(i) is a Σ'-term;

– if $t = \omega$ for some constant $\omega : \longrightarrow s \in \Omega$ then, by Definition 21(ii) and Definition 20(ii), we directly have $\mu_T(t) = \omega' \in T_{\Sigma'}X'$ for some $\omega' \in \mathrm{Op}(\Sigma')$; and

– if $t = \omega(t_1, \ldots, t_n)$ then using the induction hypothesis, $\mu_T(t_1), \ldots, \mu_T(t_n) \in T_{\Sigma'}X'$ and thus, by as Definition 21(iii) and Definition 20(iii), it is the case that

$$\mu_T(t) = \omega'(\mu_T(t_1), \ldots, \mu_T(t_n)) \in T_{\Sigma'}X'$$

for some $\omega' \in \mathrm{Op}(\Sigma')$.

These are all possible cases and we have therefore shown that the proposition claim is correct. ■

The idea of term morphisms may easily be illustrated by a simple example such as the following.

Example 19 Let $\Sigma_{\mathrm{NAT}'}$ and $\mu : \Sigma_{\mathrm{NAT}'} \longrightarrow \Sigma_{\mathrm{NATBOOL}}$ be as defined in Example 17. Consider the $\Sigma_{\mathrm{NAT}'}$-term $Succ(Add(x, Succ(Zero)))$ where $x \in X_{natural}$ is a variable belonging to the family of variables X which, in turn, is associated to $\Sigma_{\mathrm{NAT}'}$. Translating this term into a $\Sigma_{\mathrm{NATBOOL}}$ one is then done by applying the rules of Definition 21 as follows:

$$\begin{aligned}\mu(Succ(Add(x, Succ(Zero)))) &= Succ(\mu(Add(x, Succ(Zero))))\\ &= Succ(+(\mu(x), \mu(Succ(Zero))))\\ &= Succ(+(y, Succ(\mu(Zero))))\\ &= Succ(+(y, Succ(0)))\end{aligned}$$

where $\mu_X(x) = y \in Y_{nat}$ and Y is a family of variables associated to $\Sigma_{\mathrm{NATBOOL}}$. □

Finally, since each variable in a term is given a sort and, similarly, each term has a sort, it is possible to formally define the process of *substituting* variables by terms. While not directly useful to us at this point, the concept of substitutions will prove to be of fundamental importance in the next chapter where we consider logics.

Definition 22 (Substitutions) Given a signature $\Sigma = (S, \Omega)$ with associated families of variables X and Y, a *substitution* $\sigma : X \longrightarrow T_\Sigma Y$ is a family of functions $\sigma = (\sigma_s)_{s \in S}$ where

$$\sigma_s : X_s \longrightarrow T_{\Sigma,s}Y$$

for each $s \in S$. Application of σ to a term $t \in T_\Sigma X$ is denoted $t\sigma$ and is defined such that

(i) if $t = x$ for $x \in X_s$, then $t\sigma = \sigma_s(x)$;

(ii) if $t = \omega$ for $\omega : \longrightarrow s \in \Omega$, then $t\sigma = t$; and

(iii) if $t = \omega(t_1, \ldots, t_n)$ for $\omega : s_1 \times \ldots \times s_n \longrightarrow s \in \Omega$, $n \geq 1$, $t_i \in T_{\Sigma,s_i}X$, and $1 \leq i \leq n$, then $t\sigma = \omega(t_1\sigma, \ldots, t_n\sigma)$. □

Example 20 Reusing the setup from Example 19 we see that given a substitution $\sigma : X \longrightarrow T_\Sigma X$ such that $\sigma_{natural}(x) = Succ(x)$ then the term $(Succ(Add(x, Succ(Zero))))\sigma$ becomes $(Succ(Add(Succ(x), Succ(Zero))))$. □

3.3 Algebras

Up to this point, we have worked only on syntactic objects which themselves have no semantic meaning. By applying an algebra onto the previously defined signatures and terms, the semantic meaning will come to light. The ultimate goal in this section is to—in a fashion similar to Section 3.1—establish a category of algebras. A number of underlying concepts have to be presented first, however. Beginning with a formal definition of algebra.

Definition 23 (Algebras) A Σ-*algebra*—or simply *algebra*—A for some signature $\Sigma = (S, \Omega)$ is a structure containing

– for each $s \in S$, a so-called *carrier set* $A(s)$;

– for each $\omega : \longrightarrow s \in \Omega$, an element $A(\omega) \in A(s)$; and

– for each $\omega : s_1 \times \ldots \times s_n \longrightarrow s \in \Omega$ with $n \geq 1$, a function

$$A(\omega : s_1 \times \ldots \times s_n \longrightarrow s) : A(s_1) \times \ldots \times A(s_n) \longrightarrow A(s). \qquad \square$$

Remark 10 It should be noted that this text deals exclusively with complete algebras, that is, algebras which do not have any partial functions associated with a operation. While partial algebras can be very useful they also require complexity beyond the scope of this thesis. For example, the intuitive algebra to the signature $\Sigma = (S, \Omega)$ where

$$\begin{aligned} S &\supseteq \{real\} \text{ and} \\ \Omega &\supseteq \{\div : real \times real \longrightarrow real, \\ &\qquad 0 : \longrightarrow real\} \end{aligned}$$

would by necessity be partial since $t \div 0$ certainly is a valid term but its value is undefined for any term t. A similar problem arise when one considers the NATLIST signature given earlier where, in its intuitive algebra, the term $Head(Nil)$ is undefined.

Fortunately, it is in many cases possible to avoid the problem altogether by introducing additional sorts. Consider, for example, the corresponding intuitive algebra to the signature $\Sigma' = (S', \Omega')$ where

$$\begin{aligned} S' &\supseteq \{real, real\_without\_zero\} \text{ and} \\ \Omega' &\supseteq \{\div : real \times real\_without\_zero \longrightarrow real, \\ &\qquad 0 : \longrightarrow real\}. \end{aligned}$$

This construction would be able to work around the problem since $t \div 0$ is not a valid term. Subsequently the algebra would remain total.

This is not a very elegant solution and is not very practical in real applications. Fortunately, more elegant solutions to this problem exist and one which has gained support—and which has found practical use in, for example, variants of the `OBJ` language family—is to take advantage of so-called *order-sorted* algebras [19, 20].

We will not go delve deeper into this issue though and for reasons of convenience, we will simply work around any problems from now on. That the problem exists should, however, be kept in mind. $\square$

Interestingly, this uncomplicated definition of algebras provide all machinery required to give meaning to the sorts and operations of a signature. The following example illustrates how this construction works.

Example 21 Given the signature Σ_{NAT} as described in Example 12, we can construct a straight-forward algebra A for it by assigning

$$\begin{aligned} A(nat) &= \mathbb{N}, \\ A(0 : \longrightarrow nat) &= 0, \text{ and} \\ A(Succ : nat \longrightarrow nat)(n) &= n+1. \end{aligned}$$

Note that in $A(0 : \longrightarrow nat) = 0$, the symbol 0 have different meaning in its two occurrences. The first occurrence is as part of a syntactic object, that is $0 \in \Omega_{\mathrm{NAT}}$, while the second is as the well-known natural number zero, that is $0 \in \mathbb{N}$. It is of course important to keep this distinction in mind so as to avoid confusion.

The previous algebra is certainly not the only algebra which is possible to apply to Σ_{NAT}. We may, for example, construct the equally valid alternative Σ_{NAT}-algebra B by assigning

$$\begin{aligned} B(nat) &= \{0,1\}, \\ B(0) &= 0, \text{ and} \\ B(Succ)(n) &= n+1 \mod 2. \end{aligned}$$

While this algebra does not match the intuitive semantics of the signature it is, as noted, just as valid. Also, for simplicity, here the operations have been abbreviated, that is, $0 : \longrightarrow nat$ and $Succ : nat \longrightarrow nat$ are written 0 and $Succ$, respectively. □

Of course, the other example signatures given earlier can be similarly "implemented" by assigning them various algebras. A final example will show the intuitive algebra for the boolean signature given in Example 13.

Example 22 The common boolean algebra, here called A_{BOOL}, over the signature Σ_{BOOL} is constructed by letting

$$A_{\mathrm{BOOL}}(bool) = \{true, false\}$$

for the sort *bool* and

$$\begin{aligned} A_{\mathrm{BOOL}}(True) &= true, \\ A_{\mathrm{BOOL}}(False) &= false, \\ A_{\mathrm{BOOL}}(\neg)(v) &= \begin{cases} true & \text{if } v = false \\ false & \text{if } v = true \end{cases}, \text{ and} \\ A_{\mathrm{BOOL}}(\wedge)(v,u) &= \begin{cases} true & \text{if } v = true \text{ and } u = true \\ false & \text{otherwise} \end{cases} \end{aligned}$$

for the operations. □

To be of practical use, the notion of an algebra must be related to the notion of terms. Binding these two concepts together allow us to describe the value of a term using the semantics provided by an algebra. This is—as can be imagined and will also later be illustrated—quite useful. Since the notion of terms as described in Definition 20 take variables into account, we must begin this examination by considering how to handle variables. In particular, we must consider how variables are assigned values. This is done using a particular function, called a *variable assignment*.

Definition 24 (Variable assignments) If $\Sigma = (S, \Omega)$ is a many-sorted signature with an associated family of variables X, and A being an algebra over Σ. Then, variable assignment is performed using a family of functions $\mathfrak{v} : X \longrightarrow A$ such that for each $s \in S$, there is a function $\mathfrak{v}_s : X_s \longrightarrow A(s)$ which assigns to each variable of X an element of the appropriate carrier set in A. □

Now terms, as they were previously defined can be given semantic meaning. This is done by establishing the rules necessary to inductively evaluate a term using a given algebra and variable assignment.

Definition 25 (Term evaluation) Let $t \in T_\Sigma X$ be a term for some signature $\Sigma = (S, \Omega)$ having X as associated family of variables. Then the *value* of t in some Σ-algebra A together with the variable assignment $\mathfrak{v} : X \longrightarrow A$ is denoted $A(\mathfrak{v})(t)$ and is defined inductively such that

$$A(\mathfrak{v})(t) = \begin{cases} \mathfrak{v}_s(x) & \text{if } t = x \in X_s \\ A(\omega) & \text{if } t = \omega, \omega : \longrightarrow s \in \Omega \\ A(\omega)(A(\mathfrak{v})(t_1), \ldots, A(\mathfrak{v})(t_n)) & \text{if } t = \omega(t_1, \ldots, t_n), n \geq 1, \\ & \quad t_i \in T_{\Sigma, s_i} X, 1 \leq i \leq n, \text{ and} \\ & \quad \omega : s_1 \times \ldots \times s_n \longrightarrow s \in \Omega \end{cases}$$

for each $s \in S$. □

Example 23 As an illustration of the inner workings of the previous definition, we may consider the Σ_{BOOL}-algebra A_{BOOL} from Example 22. In the name of brevity, we let $A = A_{\text{BOOL}}$ in this example. Also let X be an associated set of variables with $X = \{\{x, y\}_{bool}\}$ and, finally, let $\mathfrak{v} : X \longrightarrow A$ be a variable assignment such that

$$\mathfrak{v}_{bool}(x) = true$$
$$\mathfrak{v}_{bool}(y) = false.$$

Then for the term t given in Eq. (3.1), that is,

$$t = \wedge(\wedge(x, False), \neg(y)),$$

the value $A(\mathfrak{v})(t)$ is evaluated such that

$$\begin{aligned} A(\mathfrak{v})(t) &= A(\wedge)(A(\wedge)(\mathfrak{v}(x), A(False)), A(\neg)(\mathfrak{v}(y)) \\ &= A(\wedge)(A(\wedge)(true, false), A(\neg)(false)) \\ &= A(\wedge)(false, true) \\ &= false \end{aligned}$$

when A and $\mathfrak{v}$ is applied. □

It has already been mentioned that the description of a category of algebras will be given. To do this, it is required to somehow move from one algebra to another. This is done using an *algebra homomorphism*—or simply *homomorphism*.

Definition 26 (Algebra homomorphisms) If A and B are algebras for the signature $\Sigma = (S, \Omega)$, then an algebra homomorphism, $h : A \longrightarrow B$, is a family of functions h such that $h_s : A(s) \longrightarrow B(s)$ for each $s \in S$. These functions must be constructed in such a way that for any $\omega : s_1 \times \ldots \times s_n \longrightarrow s \in \Omega$, the *homomorphism condition*

$$h_s(A(\omega)(a_1, \ldots, a_n)) = B(\omega)(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$$

holds for all $a_1 \in A(s_1), \ldots, a_n \in A(s_n)$, $n \geq 0$. In particular, the case where ω is a constant gives $h_s(A(\omega)) = B(\omega)$. □

Example 24 (The identity homomorphism) The definition of an identity homomorphism to a Σ-algebra A is rather trivial. Let id_A be the identity homomorphism, then for any sort s in Σ let $\mathrm{id}_{A_s}(a) = a$ where $a \in A(s)$. □

Example 25 In Example 21 the Σ_{NAT} algebra A was constructed such that $A(nat) = \mathbb{N}$, that is, $A(nat) = \{0, 1, 2, \ldots\}$. Also, recall from the same example that $A(0) = 0$ and $A(Succ)(n) = n + 1$. Now, let A' be the Σ_{NAT}-algebra

$$\begin{aligned} A'(nat) &= \{0, 2, 4, \ldots\}, \\ A'(0) &= 0, \text{ and} \\ A'(Succ)(n) &= n + 2. \end{aligned}$$

The homomorphism $h : A \longrightarrow A'$, taking the algebra A to A', is then defined by letting

$$h_{nat}(a) = 2a.$$

That the homomorphism condition is upheld for this particular homomorphism can easily be verified by checking each Σ_{NAT}-operation; for $0 : \longrightarrow nat$ we, of course, have

$$h_{nat}(A(0)) = h_{nat}(0) = 2 \cdot 0 = 0 = A'(0)$$

and, similarly, for $Succ : nat \longrightarrow nat$, we have

$$\begin{aligned} h_{nat}(A(Succ)(n)) &= h_{nat}(n+1) \\ &= 2(n+1) \\ &= 2n+2 \\ &= A'(Succ)(2n) \\ &= A'(Succ)(h_{nat}(n)). \end{aligned}$$

Thus, the homomorphism condition is respected and h truly is a homomorphism from A to A'. □

The homomorphisms presented in the previous two examples are special in the regard that it is possible to construct an inverse homomorphism, say h^{-1}, such that $h^{-1}(h(A)) = A$. The situation is the same as described in Definition 7(i) regarding morphisms and indeed, a homomorphism with this property is similarly called an isomorphism. The underlying requirement for a homomorphism, $h : A \longrightarrow B$, to be isomorphic is that each of its h_s-functions is bijective—which clearly is the case with h_{nat} as given in Example 25.

Definition 27 (Composition of algebra homomorphisms) Let $h : A \longrightarrow B$ and $g : B \longrightarrow C$ be two homomorphisms for the Σ-algebras A, B, and C where $\Sigma = (S, \Omega)$. Then the composition $g \circ h$ is a homomorphism $g \circ h : A \longrightarrow C$ forming a family of functions with $g_s \circ h_s$ for each $s \in S$. That is, composition is performed "sort-wise". □

It still remains to show that this composition truly is a homomorphism, the following proposition establishes this claim.

Proposition 5 (A composed homomorphism is itself a homomorphism) *If g and h are homomorphisms as given in Definition 27. Then the composition $g \circ h$, also as given in Definition 27, is a homomorphism.* $\square$

PROOF The proof is performed by showing that $g_s \circ h_s$ satisfy the homomorphism condition for any sort s in Σ. To do this, let $\omega : s_1 \times \ldots \times s_n \longrightarrow s$ be an arbitrary operation in Σ. Then

$$\begin{aligned}(g_s \circ h_s)(A(\omega)(a_1, \ldots, a_n)) &= g_s(h_s(A(\omega)(a_1, \ldots, a_n))) \\ &= g_s(B(\omega)(h_s(a_1), \ldots, h_s(a_n))) \\ &= C(\omega)(g_s(h_s(a_1)), \ldots, g_s(h_s(a_n))) \\ &= C(\omega)((g_s \circ h_s)(a_1), \ldots, (g_s \circ h_s)(a_n))\end{aligned}$$

and the homomorphism condition therefore holds and subsequently $g \circ h$ is a homomorphism. $\blacksquare$

Proposition 6 (Composition of homomorphisms is associative) *Let $h : A \longrightarrow B$, $g : B \longrightarrow C$, and $k : C \longrightarrow D$ be homomorphisms for the Σ-algebras A, B, C, and D with $\Sigma = (S, \Omega)$. Then*

$$k \circ (g \circ h) = (k \circ g) \circ h,$$

that is, homomorphism composition is associative. $\square$

PROOF As composition is done sort-wise, we have, from the definition of homomorphisms, the case that for each sort s, there is a function $k_s \circ (g_s \circ h_s)$ in $k \circ (g \circ h)$. As each h_s, g_s, and k_s is a function—and regular function composition is of course associative—it must be that

$$k_s \circ (g_s \circ h_s) = (k_s \circ g_s) \circ h_s,$$

the right-hand side of which is exactly the form of the functions found in the homomorphism composition $(k \circ g) \circ h$. As the underlying functions are equal, the homomorphisms must be equal and thus homomorphism composition is associative. $\blacksquare$

Enough concepts have now been presented to, for some signature Σ, formulate a definition of the category of all Σ-algebras.

Definition 28 (The category of algebras) The category of Σ-algebras $\mathtt{Alg}_\Sigma$, for some signature Σ, is defined by letting

- $\mathrm{Ob}(\mathtt{Alg}_\Sigma)$ be the set of all algebras as presented in Definition 23, and
- for each algebra $A, B \in \mathrm{Ob}(\mathtt{Alg}_\Sigma)$, $\mathrm{Hom}_{\mathtt{Alg}_\Sigma}(A, B)$ is the set of all algebra homomorphisms from A to B. In particular, the identity morphism for an algebra $A \in \mathrm{Ob}(\mathtt{Alg}_\Sigma)$ is id_A as defined in Example 24. $\square$

Definition 28 concludes this short introduction to universal algebra. In the following chapter covering logic, we shall see that universal algebra forms a crucial component in the description of the syntax and semantics of the logics which we will work with.

Chapter 4

Logic Using Categories

In the previous chapter on universal algebra we saw that it is possible to find the value of a term given some signature, algebra, and variable assignment. This can certainly be useful of itself but being able to, for some given signature and algebra, answer questions such as

> Will $1+x \leq 2+x$ hold for **all** assignments of x?

or

> Is it **always** true that $Head(Tail(Cons(x,Cons(y,Nil)))) = Head(Cons(y,z))$?

could prove to be an even greater tool. In other words, we wish to add *logic* to our notion of universal algebra. In this chapter, the mathematical framework needed to express these notions will be developed. Since the framework will be developed using category theory as foundation we will gain a number of interesting abilities "for free". As will be seen, one of the greatest such ability will be the possibility to move from one representation of logic to another in a consistent manner!

Before moving further into the area of logic, it is possible to directly conclude that the notions presented in in the previous chapter regarding universal algebra will become an important building block when forming a description of logics. For example, the simple boolean signature and algebra presented in respectively Example 13 and Example 22 are familiar to anyone who has had even the slightest contact with mathematical logic. However, we will fortunately not be limited to the regular boolean signature and algebra.

Traditionally, logics have been expressed mainly using unsorted signatures and algebras. Further, one has often limited the logics by fixing the signature and algebra. These limitations will, following the example of [27, 17], be done away with in this chapter and both many-sorted signatures and algebras as well as signature morphisms and algebra homomorphism will be taken into account. In the chapter we will look at both semantic implication—which many will recognize from the $\models$ symbol—and syntactic implication—similarly recognizable by the $\vdash$ symbol. Their relationship with each other will be established and we will further see how to preserve their meaning while moving between logics.

Leaving syntactic implication aside for the moment, we will first construct a mathematical structure which allows the semantic implication aspect of a logic to be examined in detail. Later, we will introduce a similar structure for syntactic implication which may then be combined with the notion of institution to form a logic.

4.1 Institutions

Informally an institution establishes the relationship between signatures—or rather the strings which may be formed using a signature—and semantic implication, a relationship which often go under the name of a *model theory* [17]. The idea of representing these aspects of logic within this type of structure stems from the work of Goguen and Burstall [31] and the notation used here will to a large part follow their notation. Note that signature in this context does not necessarily mean equational signature as it did in Chapter 3 and as will be seen we may represent signatures in a number of ways depending on purpose. Similarly, the terms presented in the previous chapter will here be referred to as *equational terms*.

Before we are able to construct concrete institutions we naturally need to establish what they actually are, the formal definition is given below and following it are some more informal explanations describing the more obscure aspects of the definition.

Definition 29 (Institutions) An *institution* $\mathscr{I}$ is a quadruple $\mathscr{I} = (\texttt{Sign}, \mathit{Sen}, \mathit{Mod}, \models)$ in which

- $\texttt{Sign}$ is a category of signatures;
- $\texttt{Sign} \xrightarrow{Sen} \texttt{Set}$ is a functor where each element of $\mathit{Sen}(\Sigma)$ is called a Σ-*sentence* or Σ-*formula*;
- $\texttt{Sign} \xrightarrow{Mod} \texttt{Cat}^{op}$ is a functor where each category of $\mathit{Mod}(\Sigma)$ represents the category of Σ-*models*; and
- $\models$ is a family of *satisfaction relations* such that

  $$\models_\Sigma \ \subseteq \mathrm{Ob}(\mathit{Mod}(\Sigma)) \times \mathit{Sen}(\Sigma)$$

  for each $\Sigma \in \mathrm{Ob}(\texttt{Sign})$.

For notational convenience, we write $M \models_\Sigma \varphi$ if $(M, \varphi) \in \ \models_\Sigma$, and $M \not\models_\Sigma \varphi$ if $(M, \varphi) \notin \ \models_\Sigma$ where $M \in \mathrm{Ob}(\mathit{Mod}(\Sigma))$ and $\varphi \in \mathit{Sen}(\Sigma)$.

Each of the $\models_\Sigma$ relations must fulfill the *satisfaction condition*—also known as $\models$-*invariance*. This condition states that for all morphisms $\mu \in \mathrm{Hom}_{\texttt{Sign}}(\Sigma, \Sigma')$, models $M' \in \mathrm{Ob}(\mathit{Mod}(\Sigma))$ and sentences $\varphi \in \mathit{Sen}(\Sigma)$, $\models_\Sigma$ must be such that

$$\mathit{Mod}(\mu)(M') \models_\Sigma \varphi \iff M' \models_{\Sigma'} \mathit{Sen}(\mu)(\varphi). \tag{4.1}$$

□

In Definition 29, note in particular that the *Mod* functor translates categories of models in the opposite direction in relation to the way signatures are translated. The rationale behind this definition of *Mod*—which at first glance may seem peculiar—may best be illustrated by considering what would happen if the seemingly more natural functor $\mathit{Mod}' : \texttt{Sign} \longrightarrow \texttt{Cat}$ had been chosen over $\mathit{Mod} : \texttt{Sign} \longrightarrow \texttt{Cat}^{op}$ in the definition of institution. The satisfaction condition would then have been

$$M \models_\Sigma \varphi \iff \mathit{Mod}'(\mu)(M) \models_{\Sigma'} \mathit{Sen}(\mu)(\varphi).$$

for a model $M \in \mathrm{Ob}(\mathit{Mod}'(\Sigma))$. Now consider the situation that occurs for the signature morphisms $\Sigma_1 \xrightarrow{\mu_1} \Sigma \xleftarrow{\mu_2} \Sigma_2$. That is, μ_1 and μ_2 have the same codomain but different domains. It is now possible to see how a situation may arise where $M_1 \in \mathrm{Ob}(Mod'(\Sigma_1))$, $\varphi_1 \in Sen(\Sigma_1)$, $M_2 \in \mathrm{Ob}(Mod'(\Sigma_2))$, and $\varphi_2 \in Sen(\Sigma_2)$ are such that $Mod'(\mu_1)(M_1) = Mod'(\mu_2)(M_2) = M$ and $Sen(\mu_1)(\varphi_1) = Sen(\mu_2)(\varphi_2) = \varphi$. But then, if $M_1 \models \varphi_1$ and $M_2 \not\models \varphi_2$, what about $M \models \varphi$? In other words, we have discovered an inconsistency. The *Mod* functor solves this problem by effectively reversing the signature morphisms. This may be illustrated by reusing the setup which caused the problem for *Mod*$'$. We then observe that $Mod(\mu_1)(M) = Mod(\mu_2)(M)$ only if $\mu_1 = \mu_2$, that is, the confusing situation where two different signature morphisms will yield the same model is avoided.

In conclusion, the *Mod* functor as given in Definition 29 is therefore the natural choice for our notion of institution and the satisfaction condition must then be as shown in Figure 4.1. The *Sen* functor is far more straight-forward and does not pose

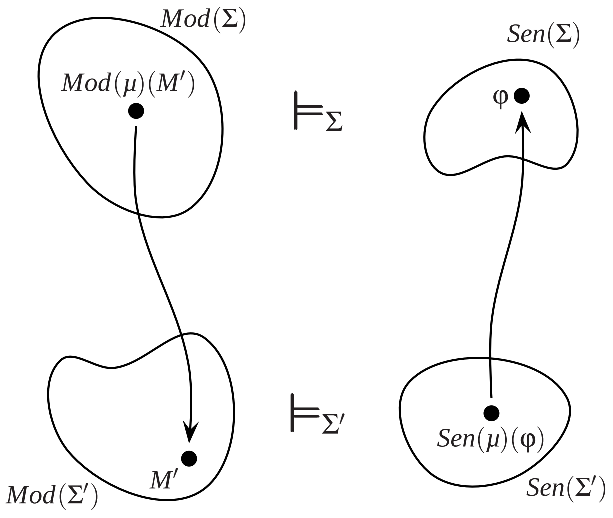

Figure 4.1: Illustration of the satisfaction condition.

any problems since we clearly want sentences to map in the same direction as signatures. The opposite would certainly be counter-intuitive! Considering the satisfaction condition further, we see that it gives as a consequence the property that satisfiability of a sentence does not change with the underlying signature representation. In short, it does not matter what name an operation may go under, it is the *operation semantics* which is of primary importance in this case.

We often wish to view the satisfaction relation not in terms of the more abstract notion of models that we have considered up to now but rather from the point of view of the set of sentences satisfied by these models. A set, say Γ, of this type for some model $M \in Mod(\Sigma)$, $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$, is determined by

$$\Gamma = \{\varphi \in Sen(\Sigma) \mid M \models_\Sigma \varphi\}.$$

From this type of representation, we are able to construct a full subcategory of $Mod(\Sigma)$ which shall be called $Mod(\Sigma, \Gamma)$. This subcategory has as objects all models which satisfy the sentences in Γ. This in turn allow us to define a slightly different family of satisfaction relations.

Definition 30 (An alternative satisfaction relation) Let $\mathscr{I} = (\texttt{Sign}, Sen, Mod, \models)$ be an arbitrary institution and $\models$ be a family of binary relations where

$$\models_\Sigma \subseteq \mathcal{P}Sen(\Sigma) \times Sen(\Sigma)$$

for each signature $\Sigma \in \text{Ob}(\texttt{Sign})$ and where $\mathcal{P}$ is the regular powerset functor. Then each $\models_\Sigma$ is also said to be a satisfaction relation if the condition

$$\Gamma \models_\Sigma \varphi \iff M \models_\Sigma \varphi$$

holds for all $\varphi \in Sen(\Sigma)$ and $M \in Mod(\Sigma, \Gamma)$. □

Remark 11 Note that in most other texts, no typographic distinction is made between the two types of satisfaction relations. In other words, $\models$ and $\models$ as previously defined are both represented by the single symbol $\models$. The reader of those texts should keep in mind that these are two distinct, albeit related, relations. The decision to represent these using two different symbols in this text was made in an attempt to highlight this distinction. □

We have in Definition 29 established what the objects in the category of institutions are, the definition of its morphisms will prove more tricky and they will therefore be omitted until the required notions have been presented. The impatient reader may find the presentation of institution morphisms in Section 4.6. The rest of this section will instead thoroughly demonstrate two motivating examples which show how the semantic part of concrete logics may be described in the context of institutions. These examples will cover equational logic and first order logic, respectively. Typically, these logics are considered in their regular unsorted, fixed signature variants but here they will be allowed the freedom of both many-sorted and varying signatures.

4.1.1 Institution of Equational Logic

In the introduction to the previous chapter it was mentioned that care should be taken to differentiate between abstract algebra and elementary algebra. Abstract algebra is a considerably more general concept and we will now use the tools developed in the chapter to describe a logic which—given an appropriate signature and algebra—may appear very much like the familiar elementary algebra.

As a first step towards this definition, it is beneficial to establish what will be considered a sentence in equational logic and how to give it an appropriate functorial representation. In other words, how should *Sen* in this particular logic be defined? Informally a sentence in equational logic may be thought of as being the same thing as equations in elementary algebra. That is, we are simply talking about *equality* and wish to answer questions of the type: Does *something* equals *something else*? Of course, the answer to such a question depend intimately on the nature of the particular algebras used. As might be suspected, a sentence in the equational logic setting will also be called an *equation* and Definition 31 that follows provide the formal description.

Definition 31 (Equations) Let e be a Σ-*equation* for some signature $\Sigma = (S, \Omega)$. The equation is represented as a triple $e = (X, t, t')$ where X is a family of Σ-variables and $t, t' \in T_{\Sigma,s}X$, that is, terms of some sort $s \in S$. For simplicity, the triple will typically be written $(\forall X)\ t = t'$. If $X = \text{var}(t) \cup \text{var}(t')$, we may simplify even further by writing $t = t'$. □

Note that the signatures in equational logic are, as the name suggest, identical to the equational signatures of the previous chapter and subsequently the `Sign` category of the institution will be the previously defined category of equational signatures, $\mathtt{Sign}^{Eq}$.

The inclusion of the family of variables—X in this case—in an equation warrants a brief explanation: In short, in unsorted equational logic an equation is typically represented simply as a pair (t,t'), where t,t' are terms. It is then possible to apply the common inference (or deduction) rules that apply in equational logic—such as reflexivity, symmetry, and transitivity. It is desirable to maintain the same inference rules even in the many-sorted case. Unfortunately, simply reusing the unsorted representation of an equation will compromise the soundness of the many-sorted logic. The underlying reasons for this will not be further developed in this text and suffice it to say, a few different solutions to this problem has been suggested but one that has gained considerable traction requires a representation of equations equivalent to the one given in Definition 31. For more information, the interested reader is referred to e.g. [18] where the problem and solutions are presented in greater detail.

No semantic meaning is placed into an equation at this point, the meaning will be provided by the satisfaction relation given in Definition 34. Fortunately, the lack of semantics does not prevent us from constructing a few simple example equations before then.

Example 26 (Simple equations) Using the signature Σ_{BOOL} from Example 13 and boolean variables b,b', we may for example construct the equations

$$(\forall\{\{b\}_{bool}\})\ b\wedge b = True \wedge b,$$
$$(\forall\{\{b,b'\}_{bool}\})\ b\wedge b' = b'\wedge b, \text{ and}$$
$$(\forall\{\{b\}_{bool}\})\ \neg b\wedge b = b.$$

Similarly, using signature $\Sigma_{\mathrm{NATLIST}}$ from Example 15, *nat*-variable x, and *bool*-variable b, we may for example construct the equations

$$(\forall\{\{x\}_{nat}\})\ Cons(x,Nil) = Nil, \text{ and}$$
$$(\forall\{\{x\}_{nat},\{b\}_{bool}\})\ b = Empty(Cons(x,Cons(0,Nil))). \qquad \square$$

It is now possible to create an appropriate *Sen* functor for our institution, taking a signature to its set of sentences—or as they are known in this case; equations. This functor will in equational logic be called Sen^{Eq} and the following definition describes its structure.

Definition 32 (The Sen^{Eq} functor) Let $Sen^{Eq} : \mathtt{Sign}^{Eq} \longrightarrow \mathtt{Set}$ be a functor such that

$$Sen^{Eq}(\Sigma) = \{(X,t,t') \mid t,t' \in T_{\Sigma,s}X, s\in S, \text{ and } X \text{ a family of variables for } \Sigma\}$$

for any signature $\Sigma = (S,\Omega) \in \mathrm{Ob}(\mathtt{Sign}^{Eq})$, and

$$Sen^{Eq}(\mu)((X,t,t')) = (\mu(X),\mu(t),\mu(t'))$$

for some signature morphism $\mu : \Sigma \longrightarrow \Sigma'$. In other words, $Sen^{Eq}(\Sigma)$ is the set of all Σ-equations and $Sen^{Eq}(\mu)(\varphi)$ turns a Σ-equation φ into a Σ'-equation. $\square$

Much like the case of signatures in equational logic, the models of equational logic are precisely the equational algebras of the previous chapter. Using the category of equational algebras which was defined there, we may construct a functor, Mod^{Eq}, taking the role of the *Mod* functor in the institution.

Definition 33 (The Mod^{Eq} functor) Let $Mod^{Eq} : \mathtt{Sign}^{Eq} \longrightarrow \mathtt{Cat}^{op}$ be a functor such that for some signature $\Sigma = (S, \Omega) \in \mathrm{Ob}(\mathtt{Sign}^{Eq})$ it is the case that

$$Mod^{Eq}(\Sigma) = \mathtt{Alg}_{\Sigma}.$$

Additionally, for some signature morphism $\mu : \Sigma \longrightarrow \Sigma'$ where $\Sigma = (S, \Omega) \in \mathrm{Ob}(\mathtt{Sign}^{Eq})$ it is the case that $Mod^{Eq}(\mu)$ is a functor

$$\mathtt{Alg}_{\Sigma'} \xrightarrow{Mod^{Eq}(\mu)} \mathtt{Alg}_{\Sigma}$$

such that

(i) if A' is a Σ'-algebra, then there exists a Σ-algebra $A = Mod^{Eq}(\mu)(A')$ which satisfies the conditions

$$A(s) = A'(\mu(s)) \quad \text{and} \quad A(\omega) = A'(\mu(\omega))$$

for each sort $s \in S$ and operation $\omega \in \Omega$; and

(ii) if $h' : A' \longrightarrow B'$ is a homomorphism between the Σ'-algebras A' and B', then the homomorphism $h = Mod^{Eq}(\mu)(h')$ is such that

$$Mod^{Eq}(\mu)(A') \xrightarrow{h} Mod^{Eq}(\mu)(B')$$

and $h_s = h'_{\mu(s)}$ for all $s \in S$. □

Two simple examples will illustrate the operation of Mod^{Eq}, the first show how algebras are translated—the case in Definition 33(i)—and the second example naturally show how algebra homomorphisms are translated—as per Definition 33(ii). Notice in the examples the way in which Mod^{Eq} translate the semantics of algebras and algebra homomorphisms in reverse.

Example 27 Let $\mu : \Sigma_{\mathrm{NAT'}} \longrightarrow \Sigma_{\mathrm{NATBOOL}}$ be as given in Example 17 and

$$A' \in \mathrm{Ob}(Mod^{Eq}(\Sigma_{\mathrm{NATBOOL}}))$$

be the traditional $\Sigma_{\mathrm{NATBOOL}}$-algebra. By applying $Mod^{Eq}(\mu)$ as per Definition 33, we obtain the $\Sigma_{\mathrm{NAT'}}$-algebra $A = Mod^{Eq}(\mu)(A')$ such that

$$\begin{aligned}
&A(natural) = A'(\mu(natural)) = A'(nat) = \mathbb{N}\\
&A(Zero) = A'(\mu(Zero)) = A'(0) = 0\\
&A(Succ)(n) = A'(\mu(Succ))(n) = A'(Succ)(n) = n+1\\
&A(Add)(n_1, n_2) = A'(\mu(Add))(n_1, n_2) = A'(+)(n_1, n_2) = n_1 + n_2.
\end{aligned}$$

□

Example 28 Just as in the previous example, let $\Sigma_{\mathrm{NAT'}} \xrightarrow{\mu} \Sigma_{\mathrm{NATBOOL}}$ be as given in Example 17 and let A' be the traditional $\Sigma_{\mathrm{NATBOOL}}$-algebra. Also, let B' be the $\Sigma_{\mathrm{NATBOOL}}$-algebra obtained by the algebra homomorphism $A' \xrightarrow{h'} B'$ defined by

$$\begin{aligned}
&h'_{nat}(n) = n^2\\
&h'_{bool}(b) = \begin{cases} true & \text{if b = false,}\\ false & \text{if b = true.}\end{cases}
\end{aligned}$$

Now we may ask ourselves what $Mod^{Eq}(\mu)(h')$ is. Simply applying Definition 33(ii) gives, for the single sort *natural* in $\Sigma_{\mathrm{NAT}'}$,

$$Mod^{Eq}(\mu)(h')_{natural} = h'_{nat}.$$

Notice in particular how—since $\Sigma_{\mathrm{NAT}'}$ has no sort corresponding to the sort *bool* in $\Sigma_{\mathrm{NATBOOL}}$—the h'_{bool} function is given no attention by $Mod^{Eq}(\mu)$.

This example will not expand further on the nature of the algebras $Mod^{Eq}(\mu)(A')$ and $Mod^{Eq}(\mu)(B')$. Rather, this will be left as an interesting exercise. □

Having established precisely the structures which constitute sentences and models in equational logic, we may now attempt to formalize an appropriate satisfaction relation.

Definition 34 (The satisfaction relation) Let $A \in \mathrm{Ob}(Mod^{Eq}(\Sigma))$ be any algebra for the equational signature Σ with X being an associated family of variables. Then the satisfaction relation of equational logic, $\models^{Eq}$, is such that

$$A \models^{Eq}_{\Sigma} (\forall X)\; t = t' \iff A(\mathfrak{v})(t) = A(\mathfrak{v})(t')$$

for each equation $(\forall X)\; t = t'$ and variable assignment $\mathfrak{v} : X \longrightarrow A$. □

It remains to show that this definition of the satisfaction relation fulfill the satisfaction condition of Eq. (4.1). This is an absolute requirement for an institution so the proof will be covered in some detail. However, before presenting this proof a lemma is needed. It, somewhat simplified, states that for some signature morphism $\Sigma \xrightarrow{\mu} \Sigma'$ and Σ'-algebra A', the Σ-terms evaluated by the algebra $Mod^{Eq}(\mu)(A')$ will yield the same value as their corresponding Σ'-terms evaluated by the algebra A'.

Lemma 1 *Let $\Sigma = (S, \Omega)$ and Σ' be two equational signatures, $\mu : \Sigma \longrightarrow \Sigma'$ a morphism between them, and X, X' be families of variables associated with Σ and Σ', respectively. Further, let A' be a Σ'-algebra and $\mathfrak{v}' : X' \longrightarrow A'$ be a variable assignment. For some term $t \in T_\Sigma X$, it is then the case that*

$$Mod^{Eq}(\mu)(A')(\mathfrak{v})(t) = A'(\mathfrak{v}')(\mu(t)) \tag{4.2}$$

where $\mathfrak{v} : X \longrightarrow Mod^{Eq}(\mu)(A')$ is a variable assignment such that

$$\mathfrak{v}_s(x) = \mathfrak{v}'_{\mu(s)}(\mu(x)) \tag{4.3}$$

for each $x \in X_s$ and $s \in S$. □

PROOF The lemma is shown by performing induction over a term $t \in T_\Sigma X$. For brevity, let $A = Mod^{Eq}(\mu)(A')$. Then

1. if $t = x$ for some $x \in X_s$, we have

$$\begin{aligned} A(\mathfrak{v})(x) &= \mathfrak{v}_s(x) && \text{by Definition 25} \\ &= \mathfrak{v}'_{\mu(s)}(\mu(x)) && \text{by Eq. (4.3)} \\ &= A'(\mathfrak{v}')(\mu(x)) && \text{by Definition 25; and} \end{aligned}$$

2. if $t = \omega$ for some $\omega : \longrightarrow s \in \Omega$, we have

$$\begin{aligned} A(\mathfrak{v})(\omega) &= A(\omega) && \text{by Definition 25} \\ &= A'(\mu(\omega)) && \text{by Definition 33(i)} \\ &= A'(\mathfrak{v}')(\mu(\omega)) && \text{by Definition 25; and} \end{aligned}$$

3. if $t = \omega(t_1, \ldots, t_n)$ for $\omega : s_1 \times \ldots \times s_n \longrightarrow s \in \Omega$ and $t_i \in T_{\Sigma,s_i}X$, $1 \leq i \leq n$, $n \geq 1$, we have

$$\begin{aligned} &A(\mathfrak{v})(\omega(t_1, \ldots, t_n)) \\ &\quad = A(\omega)(A(\mathfrak{v})(t_1), \ldots, A(\mathfrak{v})(t_n)) && \text{by Definition 25} \\ &\quad = A'(\mu(\omega))(A'(\mathfrak{v}')(\mu(t_1)), \ldots, A'(\mathfrak{v}')(\mu(t_n))) && \text{by Definition 33(i) and} \\ & && \quad \text{induction hypothesis} \\ &\quad = A'(\mathfrak{v}')(\mu(\omega(t_1, \ldots, t_n))) && \text{by Definitions 25 and 21.} \end{aligned}$$

These are all possible cases and we may therefore conclude that the Lemma holds. ■

Using Lemma 1, we are now able to show the validity of the following proposition which establishes that the satisfaction condition is upheld by the presented constructions:

Proposition 7 *The satisfaction relation as given in Definition 34 comply with the satisfaction condition. That is,*

$$Mod^{Eq}(\mu)(A') \models^{Eq}_{\Sigma} (\forall X)\, t = t' \iff A' \models^{Eq}_{\Sigma'} Sen^{Eq}(\mu)((\forall X)\, t = t') \tag{4.4}$$

for each $\Sigma = (S, \Omega), \Sigma' \in \mathrm{Ob}(\mathtt{Sign}^{\mathtt{Eq}})$ with X a family of variables for Σ as well as each $\mu : \Sigma \longrightarrow \Sigma'$, $A' \in \mathrm{Ob}(Mod^{Eq}(\Sigma'))$, and $(\forall X)\, t = t' \in \mathrm{Ob}(Sen^{Eq}(\Sigma))$. □

PROOF We first recall from Definition 32 that

$$Sen^{Eq}(\mu)((\forall X)\, t = t') = (\forall \mu(X))\, \mu(t) = \mu(t'). \tag{4.5}$$

Now, for brevity, let $A = Mod^{Eq}(\mu)(A')$ and also let $\mathfrak{v} : X \longrightarrow A$ and $\mathfrak{v}' : \mu(X) \longrightarrow A'$ be any variable assignments such that

$$\mathfrak{v}_s(x) = \mathfrak{v}'_{\mu(s)}(\mu(x))$$

for all $x \in X_s$ and $s \in S$. Then by Definition 34 and Eq. (4.5), we may state the satisfaction condition in the following, equivalent, way:

$$A(\mathfrak{v})(t) = A(\mathfrak{v})(t') \iff A'(\mathfrak{v}')(\mu(t)) = A'(\mathfrak{v}')(\mu(t')).$$

We are now able to, using Lemma 1, directly conclude that this equivalence holds and by extension that the satisfaction condition holds. ■

A simple example of course be given to help illustrate the proposition claim.

Example 29 Let the $\Sigma_{\mathrm{NAT}'}$-algebra A, $\Sigma_{\mathrm{NATBOOL}}$-algebra A', and signature morphism μ from $\Sigma_{\mathrm{NAT}'}$ to $\Sigma_{\mathrm{NATBOOL}}$ be as given in Example 27. Does then Eq. (4.4) hold for the, obviously satisfiable, $\Sigma_{\mathrm{NAT}'}$-equation $Succ(0) = Add(Succ(0), 0)$? That is, does

$$\begin{aligned} &Mod^{Eq}(\mu)(A') \models^{Eq} Succ(0) = Add(Succ(0), 0) \\ &\qquad\qquad \iff A' \models^{Eq} Sen^{Eq}(\mu)(Succ(0) = Add(Succ(0), 0)) \end{aligned} \tag{4.6}$$

hold? We first observe that by Example 27, $A = Mod^{Eq}(\mu)(A')$ and by Definition 32

$$\begin{aligned} &Sen^{Eq}(\mu)(Succ(0) = Add(Succ(0),0)) \\ &\quad = (\mu(Succ(0)) = \mu(Add(Succ(0),0)) \\ &\quad = (Succ(0) = Succ(0)+0), \end{aligned}$$

and Eq. (4.6) can therefore be rewritten

$$A \models^{Eq} Succ(0) = Add(Succ(0),0) \iff A' \models^{Eq} Succ(0) = Succ(0)+0.$$

It is easy to see that $Succ(0) = Succ(0)+0$ is satisfied by A' which is precisely what the satisfaction condition demands. Similarly, if some false $\Sigma_{\mathrm{NAT}'}$-equation where to be used instead, the corresponding $\Sigma_{\mathrm{NAT}'}$-equation would not be satisfied by A'. □

All the necessary components for the description of the institution of equational logic have now been covered. To conclude the example, the complete institution for equational logic is given in the following definition.

Definition 35 (The institution of equational logic) Let $\mathscr{I}^{Eq}$ denote the institution describing the semantic aspect of equational logic. Then

$$\mathscr{I}^{Eq} = (\mathtt{Sign}^{Eq}, Sen^{Eq}, Mod^{Eq}, \models^{Eq})$$

with each component being as defined throughout Section 4.1.1. □

4.1.2 Institution of First Order Logic

The definition of first order logic given in this section will, as previously stated, be more general than the common definition through the addition of multiple sorts and varying signature. Another difference from the typical definition of first order logic is the addition of equality, by adding this generalization we may forgo the need for predicate symbols—these may instead be “simulated” by boolean operations. That is, instead of having a predicate $SunIsShining(x)$, we may use a similarly named boolean operation together with equality, $SunIsShining(x) = True$, to achieve the same effect. This is only a convenience and we could equally well have described first order logic using predicates—the added expense would have been the necessity for further descriptions and explanations.

In essence, the definitions given in this example follow the ones in [27] but adds categorical “wrapping” by giving the description within the context of institutions. The goal in this example will be the construction of the institution of first order logic; here denoted $\mathscr{I}^{Fo}$.

As in the case of the equational logic institution, we begin by considering the sentences which in first order logic will be called *formulas*—or more precisely *first order formulas*. Due to allowing equality we are not only able to forgo the need for explicit predicates, we are also able to reuse the equational signatures in our description of the first order formulas. That is, we have

$$\mathtt{Sign}^{Fo} = \mathtt{Sign}^{Eq}. \tag{4.7}$$

The precise appearance of these formulas will now be formalized in the construction of the first order *Sen* functor.

Definition 36 (The Sen^{Fo} functor) Let $Sen^{Fo} : \mathtt{Sign}^{Fo} \longrightarrow \mathtt{Set}$ be a functor such that, for some signature $\Sigma = (S,\Omega) \in \mathrm{Ob}(\mathtt{Sign}^{Fo})$, $Sen^{Fo}(\Sigma)$ is the set of first order formulas defined inductively[1] such that

- if $t,t' \in T_{\Sigma,s}X$ for some $s \in S$ and X is family of variables associated with Σ, then $(t = t') \in Sen^{Fo}(\Sigma)$;
- if $\varphi, \varphi' \in Sen^{Fo}(\Sigma)$, then $(\varphi \wedge \varphi) \in Sen^{Fo}(\Sigma)$;
- if $\varphi \in Sen^{Fo}(\Sigma)$, then $(\neg\, \varphi) \in Sen^{Fo}(\Sigma)$; and
- if $\varphi \in Sen^{Fo}(\Sigma)$, $x \in X_s$, and $s \in S$, then $((\forall x : s)\ \varphi) \in Sen^{Fo}(\Sigma)$.

For some signature morphism $\mu : \Sigma \longrightarrow \Sigma'$, with $\Sigma, \Sigma' \in \mathrm{Ob}(\mathtt{Sign}^{Fo})$, we define the corresponding formula morphism, $Sen^{Fo}(\mu)$, inductively by letting

$$Sen^{Fo}(\mu)(f) = \begin{cases} \mu(t) = \mu(t') & \text{if } f = (t = t') \\ Sen^{Fo}(\mu)(\varphi) \wedge Sen^{Fo}(\mu)(\varphi') & \text{if } f = (\varphi \wedge \varphi') \\ \neg\, Sen^{Fo}(\mu)(\varphi) & \text{if } f = (\neg\, \varphi) \\ (\forall \mu(x) : \mu(s))\ Sen^{Fo}(\mu)(\varphi) & \text{if } f = ((\forall x : s)\ \varphi). \end{cases}$$

□

Remark 12 As before, we use the regular operator priorities and therefore allow dropping the parentheses when appropriate. We may also define a number of common formula abbreviations in terms of $\wedge$, $\neg$, and $\forall$. For example

$$\begin{aligned} \varphi \vee \varphi' &\rightsquigarrow \neg((\neg\, \varphi) \wedge (\neg\, \varphi')), \\ \varphi \longrightarrow \varphi' &\rightsquigarrow \neg\varphi \vee \varphi', \text{ and} \\ (\exists x : s)\ \varphi &\rightsquigarrow \neg((\forall x : s)\ \neg(\varphi)). \end{aligned}$$

□

Example 30 Using the signature $\Sigma_{\mathrm{NATLIST}}$ from Example (15), we may construct valid first order formulas such as

$$(\forall x : nat)\ (\forall l : natlist)\ ((\forall y : natlist)\ Cons(x,y) = l) \wedge (Head(l) = x) \text{ or}$$
$$(\forall x : nat)\ (\exists y : nat)\ \neg(x = 0) \longrightarrow (x = Succ(y) \wedge y \leq x).$$

Expanding the abbreviations applicable to the latter of the above formulas yield the equivalent, but far less convenient, formula

$$(\forall x : nat)\ \neg((\forall y : nat)\ \neg\neg\neg(\neg\neg(x = 0) \wedge \neg(x = Succ(y) \wedge y \leq x)).$$

Finally, note that there is a considerable risk of confusion regarding the symbols used in formulas. For example, $\wedge$ is an operator in $\Sigma_{\mathrm{NATLIST}}$ while the $\boldsymbol{\wedge}$ which is used above is a *meta-operation* of first order formulas. This confusing situation will—as much as possible—be avoided in this text by letting the latter symbol be bold-face. Other texts often do not make this distinction so careful reading is advised in order to avoid getting tangled up in the meaning of some particular notation. □

[1] It is also possible to construct this definition using so-called generated algebras [17]. However, as this type of algebra has not been previously covered, the straight-forward—but perhaps more mundane—inductive definition is given instead.

In order for us to add semantic meaning to these sentences, we first need to introduce the notion of *free variables*. In the following definition, recall that $\mathrm{var}(t)$ denotes the family of variables which occur in the term t.

Definition 37 (Free variables) Let $\varphi \in Sen^{Fo}(\Sigma)$ be a first order formula based on some signature $\Sigma = (S,\Omega) \in \mathrm{Ob}(\mathtt{Sign}^{Fo})$ with associated family of variables, X. We may then define the family of free variables of φ, denoted $\mathrm{free}(\varphi)$, inductively by

$$\mathrm{free}(\varphi) = \begin{cases} \mathrm{var}(t) \cup \mathrm{var}(t') & \text{if } \varphi = (t{=}t') \\ \mathrm{free}(\varphi') \cup \mathrm{free}(\varphi'') & \text{if } \varphi = (\varphi' \wedge \varphi'') \\ \mathrm{free}(\varphi') & \text{if } \varphi = \neg\,\varphi' \\ \mathrm{free}(\varphi) = \mathrm{free}(\varphi) \backslash \{x\}_s & \text{if } \varphi = (\forall x : s)\ \varphi. \end{cases}$$

for each $t, t' \in T_\Sigma X$, $\varphi', \varphi'' \in Sen^{Fo}(\Sigma)$, and $x \in X_s$, $s \in S$. □

The only thing lacking before we are able to define the first order logic satisfaction relation is a *Mod* functor, which in this case will be called Mod^{Fo}. The appearance of this functor will reminisce in large part with the Mod^{Eq} functor. The difference being that the algebras involved must be extended to cope with the first order formulas, that is, we are required to define the value of a first order formula. This is analog to the evaluation of terms as described in Definition 25.

Definition 38 (Formula evaluation) Let $\Sigma \in \mathrm{Ob}(\mathtt{Sign}^{Fo})$ be a signature with X, an associated family of variables. Further, let A be an equational Σ-algebra, $\varphi \in Sen^{Fo}(\Sigma)$ some first order formula with $\mathrm{free}(\varphi) \subseteq X$, and, finally, $\mathfrak{v} : X \longrightarrow A$, a variable assignment. Then the *truth value* of φ under A, denoted $A(\mathfrak{v})(\varphi)$, is given by the inductive definition

$$A(\mathfrak{v})(\varphi) = true \iff \begin{cases} \varphi = (t{=}t'),\ t, t' \in T_\Sigma X, \text{ and } A(\mathfrak{v})(t) = A(\mathfrak{v})(t') \\ \varphi = \varphi' \wedge \varphi'',\ A(\mathfrak{v})(\varphi') = true, \text{ and } A(\mathfrak{v})(\varphi'') = true \\ \varphi = \neg\,\varphi' \text{ and } A(\mathfrak{v})(\varphi') = false \\ \varphi = (\forall x : s)\ \varphi' \text{ and } A(\mathfrak{v}[x \mapsto_s a])(\varphi') = true \text{ for all } a \in A(s) \end{cases}$$

where $\varphi', \varphi'' \in Sen^{Fo}(\Sigma)$ and the notation $\mathfrak{v}[x \mapsto_s a]$ means that for $x \in X_s$, $\mathfrak{v}_s(x) = a$. □

Example 31 In light of Definition 38, consider the first order $\Sigma_{\mathrm{NATBOOL}}$-formulas

$$(\forall y : nat)\ ((x + y \leq y){=}True) \tag{4.8}$$

$$x * 1{=}x + 0 \tag{4.9}$$

Now, let A be the traditional $\Sigma_{\mathrm{NATBOOL}}$-algebra. Then it is easy to convince oneself that the values of Eq. (4.8) and Eq. (4.9), respectively, are

$$\begin{aligned} A(\mathfrak{v})((\forall y : nat)\ ((x + y \leq y){=}True)) &= true && \text{only if } \mathfrak{v}(x) = 0 \\ A(\mathfrak{v})(x * 1{=}x + 0) &= true && \text{for all } \mathfrak{v}(x) \in A(nat) = \mathbb{N}. \end{aligned}$$
□

From the above argument, we see that the algebras in our notion of first order logic are identical to the equational case. The models are therefore also identical and subsequently, the Mod^{Fo} functor is simply defined in terms of Mod^{Eq}.

Definition 39 (The Mod^{Fo} functor) Let $Mod^{Fo} : \mathtt{Sign}^{Fo} \longrightarrow \mathtt{Cat}^{op}$ be a functor identical in all aspects, with the exception of its domain, to the Mod^{Eq} functor of Definition 33. □

Having a definition of Mod^{Fo}, we may finally consider the satisfaction relation of first order logic. Since all the needed groundwork has been performed, this definition becomes deceptively simple.

Definition 40 (The satisfaction relation) Let $A \in \mathrm{Ob}(Mod^{Fo}(\Sigma))$ and $\varphi \in \mathrm{Ob}(Sen^{Fo}(\Sigma))$ be an algebra and first order formula for some signature $\Sigma \in \mathrm{Ob}(\mathtt{Sign}^{Fo})$, respectively. Then—using Definition 38—the first order satisfaction relation, $\models^{Fo}$, is such that

$$A \models^{Fo}_{\Sigma} \varphi \iff A(\mathfrak{v})(\varphi) = true$$

holds for each variable assignment $\mathfrak{v} : \mathrm{free}(\varphi) \longrightarrow A$. □

Just as in equational logic, it must be shown that the satisfaction relation truly uphold the satisfaction condition of Eq. (4.1). The following proposition establishes this fact.

Proposition 8 *The satisfaction relation as given in Definition 40 comply with the satisfaction condition. That is,*

$$Mod^{Fo}(\mu)(A') \models^{Fo}_{\Sigma} \varphi \iff A' \models^{Fo}_{\Sigma'} Sen^{Fo}(\mu)(\varphi) \tag{4.10}$$

for all $\Sigma = (S,\Omega), \Sigma' \in \mathrm{Ob}(\mathtt{Sign}^{Fo})$, $\mu : \Sigma \longrightarrow \Sigma'$, $A' \in \mathrm{Ob}(Mod^{Fo}(\Sigma'))$, *and formula* $\varphi \in Sen^{Fo}(\Sigma)$. □

PROOF For brevity, let $A = Mod^{Fo}(\mu)(A')$. We may then, using Definition 40 rewrite Eq. (4.10) to the equivalent statement

$$A(\mathfrak{v})(\varphi) = true \iff A'(\mathfrak{v}')(Sen^{Fo}(\mu)(\varphi)) = true. \tag{4.11}$$

Thus, we wish to show that Eq. (4.11) holds for all variable assignments $\mathfrak{v} : X \longrightarrow A$ and $\mathfrak{v} : \mu(X) \longrightarrow A'$ with

$$\mathfrak{v}_s(x) = \mathfrak{v}'_{\mu(s)}(\mu(x)) \tag{4.12}$$

and where X is some family of variables for Σ.

In the derivations that follow, for brevity, $A(\mathfrak{v})(\varphi) = true$ will be written simply $A(\mathfrak{v})(\varphi)$. The proof technique will—yet again—be induction and using it we observe that if

(i) $\varphi = (t{=}t')$ for some $t, t' \in T_\Sigma X$, then Eq. (4.11) becomes

$$\begin{aligned}
A(\mathfrak{v})(t{=}t') &\iff A(\mathfrak{v})(t) = A(\mathfrak{v})(t') && \text{by Definition 38}\\
&\iff A'(\mathfrak{v}')(\mu(t)) = A'(\mathfrak{v}')(\mu(t')) && \text{by Lemma 1}\\
&\iff A'(\mathfrak{v}')(\mu(t){=}\mu(t')) && \text{by Definition 38}\\
&\iff A'(\mathfrak{v}')(Sen^{Fo}(\mu)(t{=}t')) && \text{by Definition 36;}
\end{aligned}$$

(ii) $\varphi = \varphi' \wedge \varphi''$ for some $\varphi', \varphi'' \in Sen^{Fo}(\Sigma)$ gives;

$$\begin{aligned}
&A(\mathfrak{v})(\varphi' \wedge \varphi'')\\
&\quad\iff A(\mathfrak{v})(\varphi') \text{ and } A(\mathfrak{v})(\varphi'') && \text{by Definition 38}\\
&\quad\iff A'(\mathfrak{v}')(Sen^{Fo}(\mu)(\varphi')) \text{ and } A'(\mathfrak{v}')(Sen^{Fo}(\mu)(\varphi'')) && \text{by induction hypothesis}\\
&\quad\iff A'(\mathfrak{v}')(Sen^{Fo}(\mu)(\varphi') \wedge Sen^{Fo}(\mu)(\varphi'')) && \text{by Definition 38}\\
&\quad\iff A'(\mathfrak{v}')(Sen^{Fo}(\mu)(\varphi' \wedge \varphi'')) && \text{by Definition 36}
\end{aligned}$$

(iii) $\varphi = \neg\varphi'$ for some $\varphi' \in Sen^{Fo}(\Sigma)$ gives

$$\begin{aligned}
A(\mathfrak{v})(\neg\varphi') &\iff A(\mathfrak{v})(\varphi') = false && \text{by Definition 38}\\
&\iff A'(\mathfrak{v}')(Sen^{Fo}(\mu)(\varphi')) = false && \text{by induction hypothesis}\\
&\iff A'(\mathfrak{v}')(\neg Sen^{Fo}(\mu)(\varphi')) && \text{by Definition 38}\\
&\iff A'(\mathfrak{v}')(Sen^{Fo}(\mu)(\neg\varphi')) && \text{by Definition 36; and}
\end{aligned}$$

(iv) $\varphi = (\forall x : s)\ \varphi'$ for some $\varphi' \in Sen^{Fo}(\Sigma)$, $x \in X_s$, and $s \in S$ gives

$$\begin{aligned}
&A(\mathfrak{v})((\forall x : s)\ \varphi')\\
&\quad\iff A(\mathfrak{v}[x \mapsto_s a])(\varphi') && \text{by Definition 38}\\
&\quad\iff A'(\mathfrak{v}'[\mu(x) \mapsto_{\mu(s)} \mu(a)])(Sen^{Fo}(\mu)(\varphi')) && \text{by Eq. (4.12) and}\\
& && \quad\text{induction hypothesis}\\
&\quad\iff A'(\mathfrak{v}')((\forall \mu(x) : \mu(s))\ Sen^{Fo}(\varphi)) && \text{by Definition 38}\\
&\quad\iff A'(\mathfrak{v}')(Sen^{Fo}((\forall x : s)\ \varphi)) && \text{by Definition 36.}
\end{aligned}$$

These are all possible cases and we may therefore draw the conclusion that Proposition 8 holds. ■

We have now constructed all components needed for the institution of first order logic and shown that they are consistent with the satisfaction condition. This institution is summarized in Definition 41 below.

Definition 41 (The institution of first order logic) Let $\mathscr{I}^{Fo}$ denote the institution describing the semantic aspect of first order logic. Then

$$\mathscr{I}^{Fo} = (\mathtt{Sign}^{Fo}, Sen^{Fo}, Mod^{Fo}, \models^{Fo})$$

with each component being as defined throughout Section 4.1.2 □

In these two examples, we have in some detail covered the creation of both the institution of equational logic and the institution of first order logic with equality. Many of the definitions given in these two examples will prove themselves useful throughout the rest of the chapter so a thorough understanding of the relevant notions is recommended before continuing with the rest of the chapter.

4.2 Entailment Systems

Having covered semantic implication and the semantic parts of logic in general, we shall now concentrate on the syntactic parts. We shall do this in the context of *entailment systems*. These will at this point be very abstract and unfortunately not allow us to expand the example of equational logic and first order logic into the area of syntactic implication. The abstraction is primarily due to the fact that the process of actually proving sentences is completely separated from the notion of entailment. In other words, an entailment system merely describes the *properties* we expect syntactic implication to have. Section 4.8 will reveal a concrete way to describe—or axiomatize—the proof process by adding the notion of *proof calculi* to our bag of tools.

But before then we will examine the entailment system, beginning with—as in the case of institutions—the formal definition.

Definition 42 (Entailment systems) An entailment system, $\mathscr{E}$, is constructed using a triple $\mathscr{E} = (\mathtt{Sign}, Sen, \vdash)$ where

- $\mathtt{Sign}$ is a category of signatures,
- Sen is a functor $Sen : \mathtt{Sign} \longrightarrow \mathtt{Set}$, and
- $\vdash$ is a family of binary relations consisting of

$$\vdash_\Sigma \subseteq \mathcal{P}Sen(\Sigma) \times Sen(\Sigma)$$

  for each signature $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$ where $\vdash_\Sigma$ is called a Σ-*entailment*

subject to the condition that each $\vdash_\Sigma$

(i) is reflexive, that is, $\{\varphi\} \vdash_\Sigma \varphi$ for each $\varphi \in Sen(\Sigma)$;

(ii) is monotone, that is, $\Gamma' \vdash_\Sigma \varphi$, for all $\Gamma' \subseteq \Gamma$ such that $\Gamma \vdash_\Sigma \varphi$;

(iii) is transitive, that is, given sentences φ_i, $i \in I$, such that $\Gamma \vdash_\Sigma \varphi_i$ and, additionally, $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash_\Sigma \psi$, then $\Gamma \vdash_\Sigma \psi$; and

(iv) is an $\vdash$-*translation*, meaning that, if $\Gamma \vdash_\Sigma \varphi$ then for all $\mu \in \mathrm{Hom}_{\mathtt{Sign}}(\Sigma, \Sigma')$, it is the case that $\mathcal{P}\mu(\Gamma) \vdash_{\Sigma'} \mu(\varphi)$.

If $\Gamma \vdash_\Sigma \varphi$, then we say that Γ is the set of *axioms* and φ *is derivable from* Γ or, alternatively, that φ *is a logical consequence of* Γ. Note that the subscript of $\vdash_\Sigma$ may be omitted when the signature is of little interest or is obvious from context. Finally, if an entailment system is such that for any $\Gamma \vdash_\Sigma \varphi$, there is a finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \vdash_\Sigma \varphi$, then the entailment system is called *compact*[2]. □

More informally, the reflexivity condition on $\vdash$ means that if we assume the validity of a sentence then we may also prove it. The monotonicity condition assure us that adding axioms only expand—or possibly leave unchanged—the set of provable sentences. The transitivity condition states that adding a derivable sentence to the set of axioms does not change which other sentences we may prove. Finally, $\vdash$-translation states that entailment remains invariant under change of signature. In other words, the conditions are such that the family of entailment relations is required to behave just as we intuitively expect it to!

Note that the structure of $\vdash$ precisely matches that of $\models$ as it was presented in Definition 30. Indeed, we may directly construct an entailment system from an institution as per the following example.

Example 32 Let $\mathscr{I} = (\mathtt{Sign}, Sen, Mod, \models)$ denote some institution. Using the $\models$ relation from Definition 30, we may construct an entailment system $\mathscr{I}^+ = (\mathtt{Sign}, Sen, \models)$. It is then trivially true that $\models$ fulfill the list of criteria posed in Definition 42 and that $\mathscr{I}^+$ is an entailment system. □

This result, albeit rather elegant, is not very useful in practice. We understand the reason behind this by considering an important motivation as to why the syntactic approach to satisfaction is of interest: It allows the possibility of mechanizing the proof process. In particular it sometimes allows the *efficient* mechanization of the proof

[2]The notion of compact entailment systems is similar to the notion of compact topological spaces in which each open cover has a finite subcover. See e.g. [22, 42].

process. Certainly a highly desirable property. However, for an entailment system constructed directly from logic semantics, as in Example 32, we have in general little hope of easily finding such an efficient mechanization, or indeed any at all. This is simply due to the possible—often unbounded—enormity of the set of axioms. For this reason, the idea of compact entailment systems is quite desirable as we are then able to at least restrict the set of axioms to be finite which simplifies the proof process somewhat.

We will occasionally consider all sentences derivable from a set of sentences, that is, their closure under entailment. This will be done frequently enough to warrant the special notation introduced in the following definition.

Definition 43 Let Γ be a set of Σ-sentences. We may from it construct the set of *theorems* provable from Γ, denoted $\Gamma^\bullet$, by letting

$$\Gamma^\bullet = \{\varphi \mid \Gamma \vdash \varphi\}.$$

□

Due to the reflexivity condition, we therefore have the relationship $\Gamma \subseteq \Gamma^\bullet \subseteq Sen(\Sigma)$ which may be illustrated in the context of a Venn diagram such as the one shown in Figure 4.2.

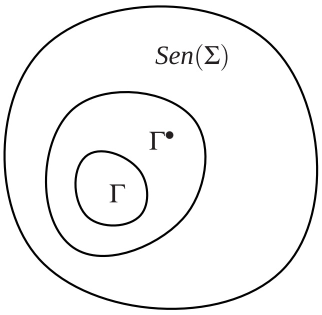

Figure 4.2: The relationship between Γ, $\Gamma^\bullet$, and $Sen(\Sigma)$.

4.3 Logics

Having defined both semantic and syntactic implication—using institutions and entailment systems, respectively—we are now able to combine these two into the description of a formal *logic*.

Definition 44 (Logics) A logic $\mathscr{L}$ is a quintuple $\mathscr{L} = (\mathtt{Sign}, Sen, Mod, \vdash, \models)$ such that

(i) $(\mathtt{Sign}, Sen, \vdash)$ is an entailment system;

(ii) $(\mathtt{Sign}, Sen, Mod, \models)$ is an institution; and

(iii) the condition

$$\Gamma \vdash_\Sigma \varphi \implies \Gamma \models_\Sigma \varphi$$

holds for all $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$, $\Gamma \in \mathcal{P}Sen(\Sigma)$, and $\varphi \in Sen(\Sigma)$.

The last condition is commonly called the *soundness condition.* If the logic also fulfill the stronger condition

$$\Gamma \vdash_\Sigma \varphi \iff \Gamma \models_\Sigma \varphi$$

then it is said to be *complete.* □

Remark 13 The notion of soundness could equivalently be stated by requiring that $\vdash_\Sigma \subseteq \models_\Sigma$ and similarly, if the logic is complete then $\vdash_\Sigma = \models_\Sigma$ holds for each signature $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$. □

Informally, the soundness condition states that it is only possible to derive things which are semantically true. This is of course an essential property for a logic to have, being able to prove things which are semantically untrue would be quite chaotic! The completeness condition adds the property that all semantically true sentences will also be true under syntactic implication. This is a desirable property but many times not entirely essential.

4.4 Theories

There are a few different—but closely related—ways of thinking about theories in this context of logic. Common to them all is that we define a theory to be a set of sentences together with the signature from which the sentences were constructed. The difference in how theories are defined stem from the way this construction is defined. The following definition—which is close to identical with the one given in [31]—will be used exclusively in this text.

Definition 45 (Theories) A *theory* for some entailment system or institution with $\mathtt{Sign}$ being its category of signatures and *Sen*, its sentence functor, is a pair $T = (\Sigma, \Gamma)$ such that

– $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$ is a signature, and

– $\Gamma \subseteq Sen(\Sigma)$ is a set of *axioms.* □

Remark 14 Recall that the notion of axioms also was introduced in the definition of entailment systems. As will become apparent, the two notions are very closely related and no confusion will arise from them both being referred to as "axioms". □

Two other definitions of theory deserve mentioning, they both let a theory be a signature combined with a set of sentences just as in Definition 45 but instead of limiting this set of sentences to just the axioms, one states that it contains sentences closed under entailment while the other states that the set of sentences consists of those which are closed under satisfaction. This warrants the claim by Meseguer that the notion of theory presented in Definition 45 could readily be called the *presentation* of these two more expansive theory concepts [31]. In defense of using Definition 45 as the basis for theories, he states:

> However, the view of theories as presentations allows us to make finer distinctions that are important for both proof-theoretic and computational purposes. We can, for example, distinguish between a sentence that is a basic axiom and another that is a derived theorem.

As will become evident, these finer distinctions prove themselves to be quite useful indeed.

To make this abstract notion of theory more concrete, we may consider the following two examples. Each example illustrates a possible theory based on signatures defined in Chapter 3.

Example 33 Let Σ_{BOOL} be the signature given in Example 13. A simple and intuitive first order theory $T_{FoBool} = (\Sigma_{\text{BOOL}}, \Gamma_{\text{FoBool}})$ might then be such that

$$\begin{aligned}
\Gamma_{\text{FoBool}} = \{ & True {=} \neg False, & (4.13)\\
& True \wedge x {=} x, & (4.14)\\
& False \wedge x {=} False & (4.15)\\
& x \wedge x {=} x \}. & (4.16)
\end{aligned}$$

□

Example 34 Let Σ_{NATLIST} be the signature given in Example 15 and X be an associated family of variables with $x \in X_{nat}$ and $y \in X_{natlist}$. Then one possible equational theory for this signature is $T_{EqNatList} = (\Sigma_{\text{NATLIST}}, \Gamma_{\text{EqNatList}})$ such that

$$\begin{aligned}
\Gamma_{\text{EqNatList}} = \{ & Empty(Nil) = True, & (4.17)\\
& Empty(Cons(x,y)) = False, & (4.18)\\
& Head(Cons(x,y)) = x, \text{ and} & (4.19)\\
& Tail(Cons(x,y)) = y \} & (4.20)
\end{aligned}$$

is the set of axioms.

Note that there is no axiom specifying the behavior of $Head(Nil)$ and $Tail(Nil)$, this relates directly to the issue discussed in Remark 10. □

Looking closer at Definition 45, we see that with the addition of morphisms, it is possible to construct a category of theories. This category will now be defined.

Definition 46 (The category of theories) Let $\mathscr{X}$ denote an institution or entailment system with signature category `Sign` and sentence functor *Sen*. Then the category of $\mathscr{X}$-theories, say `Th`, is such that $\text{Ob}(\texttt{Th})$ is the set of all $\mathscr{X}$-theories and each *theory morphism* $(\Sigma, \Gamma) \xrightarrow{\mu_{Th}} (\Sigma', \Gamma')$ with $\Sigma, \Sigma' \in \text{Ob}(\texttt{Sign})$, $\Gamma \subseteq Sen(\Sigma)$, $\Gamma' \subseteq Sen(\Sigma')$, consists of a signature morphism $\Sigma \xrightarrow{\mu} \Sigma'$ such that

$$\Gamma' \vdash_{\Sigma'} Sen(\mu)(\varphi)$$

holds for each $\varphi \in \Gamma$. If μ_{Th} also fulfills the condition that $\mathcal{P}Sen(\mu)(\Gamma) \subseteq \Gamma'$, that is, axioms are mapped to axioms, then it is said to be *axiom-preserving*. We let $\texttt{Th}_0$ denote the subcategory of `Th` containing the same objects but only axiom-preserving morphisms. □

Figure 4.3 illustrates the difference between an axiom-preserving theory morphism, μ_{Th} and a non axiom-preserving ditto, μ'_{Th}. It may be argued that the condition of axiom-preservation is quite strong, perhaps to strong, but the very nice properties it has will prove to be more valuable to us than its limitations. Besides, it is always possible to reconstruct the complete set of derivable sentences by computing the closure of the axioms under entailment.

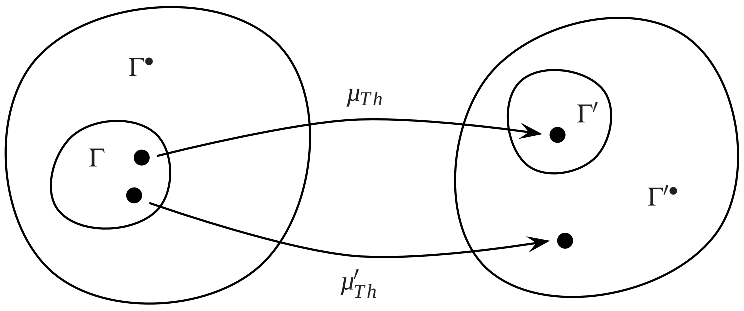

Figure 4.3: Comparison of axiom-preserving and non axiom-preserving theory morphisms, denoted μ_{Th} and μ'_{Th}, respectively

Since a theory embeds its associated signature and the set of axioms, it is possible to construct forgetful functors which given a theory extracts one or the other of these components.

Definition 47 (The $\mathcal{S}$ and $\mathcal{A}$ functors) Let $\mathcal{S} : \mathtt{Th} \longrightarrow \mathtt{Sign}$ and $\mathcal{A} : \mathtt{Th} \longrightarrow \mathtt{Set}$ be the forgetful functors taking a theory to its underlying signature and set of axioms, respectively. That is, for some theory, $T = (\Sigma, \Gamma) \in \mathrm{Ob}(\mathtt{Th})$, we have $\mathcal{S}(T) = \Sigma$ and $\mathcal{A}(T) = \Gamma$. □

Finally, we will later find use for the following simple functor which given a signature yields the theory containing that signature but holds no axioms.

Definition 48 (The $\mathcal{T}$ functor) Let $\mathcal{T} : \mathtt{Sign} \longrightarrow \mathtt{Th}$ be the functor which for some signature Σ is such that $\mathcal{T}(\Sigma) = (\Sigma, \varnothing)$. □

The notions regarding theories which have been presented throughout this section will prove themselves useful when we now move towards a description on how entailment systems and institutions may be extended to categories. We begin with the category of entailment systems as its description is somewhat less complicated. Not only is it less complicated, many of the concepts needed when describing the category of entailment systems may be directly adopted in the description of the category of institutions.

4.5 The Category of Entailment Systems

In Section 4.2, the notion of entailment systems was presented, at that point it was not yet possible to define precisely how one moves from one entailment system to another. This will now be rectified since the necessary foundations have been presented in the last few sections. Once the mapping of one entailment system to another has been covered—and we have established that these mappings are composable—it is finally possible to formally state the definition of the category of entailment systems.

Recall that, somewhat informally, an entailment system consists of a description of the syntax of a logic—in the form of signatures and sentences over them—together with an entailment relation subject to certain conditions. Considering this, the overall idea behind an map of entailment systems is then quite intuitive. The signatures and sentence of the source logic has to be mapped to valid counterparts in the target logic. This has to be done in such a way that a derivable sentence in the source logic becomes a derivable sentence in the target logic. Naturally a sentence which is not derivable in the source logic should be mapped to a similarly underivable sentence of the target logic.

Beginning with the mapping of signatures, we may directly observe that it is, of course, perfectly possible for the source and target entailment systems to use fundamentally different notions of signatures. By this is meant that the structure of their respective `Sign` categories may differ. It is therefore not simply a matter of applying our trusty old notion of signature morphisms, instead we need an appropriately defined functor mapping the `Sign` categories. For reasons which will become clear, a simple map of signature categories will not suffice. Instead, the functor will act on theories. Once the structure of this functor has been established, the mapping of sentences must follow. That is, the logics' respective *Sen* functor must be mapped in such a way that a valid sentence in the source logic is taken to a valid sentence in the target logic. Since this is a mapping of functors, a natural transformation is needed. The described functor—which shall be denoted Φ—together with the natural transformation—denoted α—will be sufficient for mapping entailment systems. As per the description above, Φ and α must be such that the signatures and sentences are translated correctly with respect to the relevant entailment relations.

We introduce the more formal treatment by a simple example, more specifically we construct a Φ and an α which takes the entailment system of equational logic to the entailment system of first-order logic.

Example 35 Let $\mathscr{E}^{Eq} = (\mathtt{Sign}^{Eq}, Sen^{Eq}, \vdash^{Eq})$ denote the entailment system for many-sorted equational logic and similarly let $\mathscr{E}^{Fo} = (\mathtt{Sign}^{Fo}, Sen^{Fo}, \vdash^{Fo})$ denote the entailment system for many-sorted first order logic. As mentioned, we must map these using a functor Φ and natural transformation α in such a way that entailment is preserved. That is, a map from $\mathscr{E}^{Eq}$ to $\mathscr{E}^{Fo}$ must have the property that for each signature $\Sigma = (S, \Omega) \in \mathrm{Ob}(\mathtt{Sign}^{Eq})$, Σ-equation $(\forall X)\; t = t'$, and set of axioms $\Gamma \in \mathcal{P}Sen^{Eq}(\Sigma)$, it is the case that

$$\Gamma \vdash^{Eq}_{\Sigma} (\forall X)\; t = t' \implies \mathcal{P}\alpha(\Gamma) \vdash^{Fo}_{\mathcal{S}\Phi(\Sigma,\varnothing)} \alpha((\forall X)\; t = t'). \tag{4.21}$$

Dissecting Eq. (4.21)—starting with the translation of equations into formulas—we observe that this is rather easy to do this using an α defined such that

$$\alpha_{\Sigma}((\forall X)\; t = t') = ((\forall x_1 : s_1)\; \ldots\; (\forall x_n : s_n)\; t{=}t')$$

where $x_i \in \mathrm{var}_{s_i}(t) \cup \mathrm{var}_{s_i}(t')$, $s_i \in S$, and $0 \leq i \leq n$. Nothing complex is occurring in this translation, the equations are simply taken to their equivalent first order formula.

Shifting attention to Φ, one may ask why it is given a theory with no axioms as input. The answer to this important question will be deferred to the discussion following this example. Before then we consider how Φ must be defined. Recall from Eq. (4.7) that the signatures of equational and first order logic are identical. In other words, the functor Φ should, given a theory, pass the signature on unaltered but change the axioms to first order axioms using the same technique as α. That is, $\Phi(\Sigma,\Gamma) = (\Sigma, \mathcal{P}\alpha_\Sigma(\Gamma))$ and $\Phi(\mu_{Th})(\Sigma,\Gamma) = (\Sigma', \mathcal{P}\alpha_{\Sigma'}(\Gamma'))$ for each theory $T = (\Sigma,\Gamma)$ and theory morphism $\mu_{Th} : T \longrightarrow T'$ where $T' = (\Sigma',\Gamma')$. In Eq. (4.21), we therefore have $\mathcal{S}\Phi(\Sigma,\varnothing) = \Sigma$.

Thus, using the familiar Σ_{NATBOOL} signature, we consider the entailment relation using the trivial but concrete equation and set of axioms

$$\{x+y=y+x, (x \le x+y) = True\} \vdash^{Eq} (j \le k+j) = True. \tag{4.22}$$

After applying the previously defined Φ and α, Eq. (4.22) becomes the following equivalent first order relation:

$$\{(\forall x : nat)\ (\forall y : nat)\ x+y=y+x, (\forall x : nat)\ (\forall y : nat)\ (x \le x+y)=True\}$$
$$\vdash^{Fo} (\forall j : nat)\ (\forall k : nat)\ (j \le k+j)=True.$$

□

The reason for Φ acting on the category of theories and not directly on signatures is that this allows far more generality and power in our notion of maps of entailment systems. While this generality was not taken advantage of in Example 35—explaining the empty set of axioms given to Φ—it is, for example, necessary in certain circumstances to add further axioms to ensure the consistency of the map. We will see examples of this shortly.

However, before then, the formal definition of maps of entailment systems will be established. Following directly thereafter is a comprehensive informal explanation of the notions involved.

Definition 49 (Map of entailment systems) Let

$$\mathcal{E} = (\mathtt{Sign}, Sen, \vdash) \text{ and } \mathcal{E}' = (\mathtt{Sign}', Sen', \vdash')$$

be two entailment systems. Then a map from $\mathcal{E}$ to $\mathcal{E}'$ is a pair $(\Phi,\alpha) : \mathcal{E} \longrightarrow \mathcal{E}'$ where

$$\mathtt{Th}_0 \xrightarrow{\Phi} \mathtt{Th}'_0$$

is a functor from the category of $\mathcal{E}$-theories to the category of $\mathcal{E}'$-theories, and

$$Sen \xrightarrow{\alpha} Sen' \circ \mathcal{S}' \circ \Phi \circ \mathcal{T}$$

is a natural transformation such that

(i) Φ maps theory signatures with no regard to axioms, that is, $\mathcal{S}'\Phi = \mathcal{S}'\Phi\mathcal{T}\mathcal{S}$;

(ii) the theories $\Phi(\Sigma,\Gamma)$ and $\Phi(\Sigma,\varnothing)$ are such that

$$(\mathcal{A}\Phi(\Sigma,\Gamma))^\bullet = (\mathcal{A}'\Phi(\Sigma,\varnothing) \cup \mathcal{P}\alpha_\Sigma(\Gamma))^\bullet; \text{ and}$$

(iii) for each $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$,

$$\Gamma \vdash_\Sigma \varphi \implies \mathcal{P}\alpha_\Sigma(\Gamma) \cup \mathcal{A}'\Phi(\Sigma,\varnothing) \vdash'_{\mathcal{S}'\Phi(\Sigma,\varnothing)} \alpha_\Sigma(\varphi). \tag{4.23}$$

We say that the map (Φ,α) is *conservative* if instead of 49(iii), the stronger condition

$$\Gamma \vdash_\Sigma \varphi \iff \mathcal{P}\alpha_\Sigma(\Gamma) \cup \mathcal{A}\Phi(\Sigma,\varnothing) \vdash'_{\mathcal{S}\Phi(\Sigma,\varnothing)} \alpha_\Sigma(\varphi)$$

holds. □

Definition 49 is not quite as complicated as it seems but care should be taken to not miss any details. Going through the conditions one by one we observe that Condition 49(i) requires the functor Φ to be such that theories with the same signature will be mapped to theories with the same signature. In other words, the output signature is uniquely defined by the input signature regardless of the particular axioms involved.

Condition 49(ii) states that the closure under entailment of the axioms translated by α together with the axioms introduced by Φ matches the corresponding closure under entailment of the axioms of the target logic. Put simply, α and Φ must be in agreement about which logic they are mapping to. A functor upholding conditions 49(i) and 49(ii) is said to be α*-sensible.*

Finally, condition 49(iii) states that Φ and α must be such that they together map signatures and sentences in a manner consistent with the logics' respective entailment relations. That is, the provability of the mapped sentence in the target logic must exactly reflect the provability of the corresponding sentence in the source logic. This condition could therefore be equivalently stated by requiring that

$$\varphi \in \Gamma^\bullet \implies \alpha_\Sigma(\varphi) \in (\mathcal{A}\Phi(\Sigma,\varnothing) \cup \mathcal{P}\alpha_\Sigma(\Gamma))^\bullet \tag{4.24}$$

holds and in practice this condition is often easier to work with—however, it is perhaps not as intuitive as the one given in Eq. (4.23).

Let us make the definition more concrete by considering a simple, but interesting, example of a map of entailment systems. This example constructs the map from an entailment system to its closure under entailment.

Example 36 Let $\mathscr{E} = (\mathtt{Sign}, \mathit{Sen}, \vdash)$ be an entailment system, we may then form a map of entailment systems $(E, \mathrm{id}_{\mathit{Sen}}) : \mathscr{E} \longrightarrow \mathscr{E}$ where $E : \mathtt{Th}_0 \longrightarrow \mathtt{Th}_0$ is a functor such that for some $\mathscr{E}$-theory $T = (\Sigma,\Gamma)$, $E(T)$ is the theory $E(T) = (\Sigma, \Gamma^\bullet)$ and $\mathrm{id}_{\mathit{Sen}}$ is the obvious identity natural transformation.

Illustrating this map in the context of Eq. (4.23) gives for each $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$

$$\Gamma \vdash_\Sigma \varphi \implies \Gamma^\bullet \vdash_\Sigma \varphi,$$

that is, the set of axioms is expanded to contain all provable sentences. It is easy to see that this map is actually conservative, that is, it is possible to change the logical implication to a logical equivalence. □

It has been shown how maps of entailment systems are constructed, it remains to show that these maps are composable. The composition given in the following definition is rather elegant in its intuitive nature.

Definition 50 (Composition of maps of entailment systems) Let $(\Phi,\alpha) : \mathscr{E} \longrightarrow \mathscr{E}'$ and $(\Phi',\alpha') : \mathscr{E}' \longrightarrow \mathscr{E}''$ be maps of entailment systems, then the composed map

$$(\Phi',\alpha') \circ (\Phi,\alpha) : \mathscr{E} \longrightarrow \mathscr{E}''$$

is defined by

$$(\Phi',\alpha') \circ (\Phi,\alpha) = (\Phi' \circ \Phi, \alpha'_{\mathcal{S}'\Phi\mathcal{T}} \circ \alpha). \tag{4.25}$$

□

Of course, we also have to show that this composition is, itself, a map of entailment systems. In order to do this we will need a simple lemma.

Lemma 2 *Let* $(\Phi,\alpha) : \mathscr{E} \longrightarrow \mathscr{E}'$*, with* $\mathscr{E} = (\mathtt{Sign}, Sen, \vdash)$*, be a map of entailment systems. Then, for each* $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$*,* $\Gamma, \Delta \in \mathcal{P}Sen(\Sigma)$*, and* $\varphi \in Sen(\Sigma)$*, the implication*

$$\Gamma \cup \Delta \vdash_\Sigma \varphi \implies \mathcal{P}\alpha_\Sigma(\Gamma) \cup \mathcal{A}\Phi(\Sigma,\Delta) \vdash'_{\mathcal{S}\Phi(\Sigma,\Delta)} \alpha_\Sigma(\varphi) \tag{4.26}$$

holds. Further, if (Φ,α) *is conservative then the two sides are equivalent.* □

PROOF Expressing Eq. (4.26) in the style of Eq. (4.24) gives

$$\varphi \in (\Gamma \cup \Delta)^\bullet \implies \alpha_\Sigma(\varphi) \in (\mathcal{A}\Phi(\Sigma,\Delta) \cup \mathcal{P}\alpha_\Sigma(\Gamma))^\bullet \tag{4.27}$$

and this is the statement we wish to prove. In essence, the result follows directly from the conditions in Definition 49. We know, since (Φ,α) is a map of entailment systems, that

$$\varphi \in (\Gamma \cup \Delta)^\bullet \implies \alpha_\Sigma(\varphi) \in (\mathcal{A}\Phi(\Sigma,\varnothing) \cup \mathcal{P}\alpha_\Sigma(\Gamma \cup \Delta))^\bullet \tag{4.28}$$

Simple derivation gives

$$\begin{aligned}
&(\mathcal{A}\Phi(\Sigma,\varnothing) \cup \mathcal{P}\alpha_\Sigma(\Gamma \cup \Delta))^\bullet \\
&\quad = \alpha_\Sigma(\varphi) \in (\mathcal{A}\Phi(\Sigma,\varnothing) \cup \mathcal{P}\alpha_\Sigma(\Gamma) \cup \mathcal{P}\alpha_\Sigma(\Delta))^\bullet \\
&\quad = \alpha_\Sigma(\varphi) \in (\mathcal{A}\Phi(\Sigma,\Delta) \cup \mathcal{P}\alpha_\Sigma(\Gamma))^\bullet && \text{by Definition 49(ii).}
\end{aligned}$$

Which when inserted in Eq. (4.28) gives exactly Eq. (4.27) which was the sentence to prove. The proof for equivalence is analogous. ■

In short, the above lemma states that we may move axioms arbitrarily between the application of α and the application of Φ with no risk of altering the entailment relation. As can be imagined, this is a very useful property and it will be taken advantage of in the proof of the proposition that follows.

Proposition 9 *The composition of entailment systems as given in Definition 50 is itself a map of entailment systems.* □

PROOF Let $(\Phi,\alpha) : \mathscr{E} \longrightarrow \mathscr{E}'$ and $(\Phi',\alpha') : \mathscr{E}' \longrightarrow \mathscr{E}''$ be maps of entailment systems and let $(\Phi^\circ,\alpha^\circ)$ denote the composition $(\Phi',\alpha') \circ (\Phi,\alpha) : \mathscr{E} \longrightarrow \mathscr{E}''$, that is, $\Phi^\circ = \Phi' \circ \Phi$ and $\alpha^\circ = \alpha'_{\mathcal{S}'\Phi\mathcal{T}} \circ \alpha$. To prove the lemma, we wish to show that the conditions of placed on maps of entailment systems are upheld. These are shown in turn.

- Condition 49(i) follows directly from the fact that Φ and Φ' follow 49(i). Thus, we get the derivation

$$\mathcal{S}''\Phi^\circ = \mathcal{S}''\Phi'\Phi = \mathcal{S}''\Phi'\mathcal{T}'\mathcal{S}'\Phi = \mathcal{S}''\Phi'\mathcal{T}'\mathcal{S}'\Phi\mathcal{T}\mathcal{S} = \mathcal{S}''\Phi'\Phi\mathcal{T}\mathcal{S} = \mathcal{S}''\Phi^\circ\mathcal{T}\mathcal{S}.$$

- Similarly, that 49(ii) holds for Φ° follows from the fact that it holds for Φ and Φ'. Let $\Phi(\Sigma,\varnothing) = (\Sigma',\varnothing')$. Using the identities $\mathcal{P}(f \circ g) = \mathcal{P}f \circ \mathcal{P}g$ and $\mathcal{S}'\Phi\mathcal{T}\Sigma = \Sigma'$ as well as Lemma 2 we find that

$$\begin{aligned}
(\mathcal{A}\Phi^\circ(\Sigma,\varnothing) \cup \mathcal{P}\alpha^\circ_\Sigma(\Gamma))^\bullet &= (\mathcal{A}\Phi'\Phi(\Sigma,\varnothing) \cup \mathcal{P}(\alpha'_{\Sigma'} \circ \alpha_\Sigma)(\Gamma))^\bullet \\
&= (\mathcal{A}\Phi'(\Sigma',\varnothing') \cup \mathcal{P}\alpha'_{\Sigma'}(\mathcal{P}\alpha_\Sigma(\Gamma)))^\bullet \\
&= (\mathcal{A}\Phi'(\Sigma',\mathcal{P}\alpha_\Sigma(\Gamma) \cup \varnothing'))^\bullet && \text{by Lemma 2} \\
&= (\mathcal{A}\Phi'\Phi(\Sigma,\Gamma))^\bullet \\
&= (\mathcal{A}\Phi^\circ(\Sigma,\Gamma))^\bullet.
\end{aligned}$$

– Condition 49(iii), when rewritten as per Eq. (4.24) becomes

$$\varphi \in \Gamma^{\bullet} \implies \alpha^{\circ}_{\Sigma}(\varphi) \in (\mathcal{A}\Phi^{\circ}(\Sigma, \varnothing) \cup \mathcal{P}\alpha^{\circ}_{\Sigma}(\Gamma))^{\bullet}.$$

Given $\Phi(\Sigma, \varnothing) = (\Sigma', \varnothing')$, then when expanded further and rewritten slightly using the same technique as above we get the equivalent statement

$$\varphi \in \Gamma^{\bullet} \implies \alpha'_{\Sigma'}(\alpha_{\Sigma}(\varphi)) \in (\mathcal{A}\Phi'(\Sigma', \varnothing) \cup \mathcal{P}\alpha'_{\Sigma'}(\mathcal{P}\alpha_{\Sigma}(\Gamma) \cup \varnothing'))^{\bullet}.$$

Since both (Φ, α) and (Φ', α') are maps of entailment systems it follows directly that this implication hold.

Thus, Φ° and α° are in compliance with the conditions of Definition 49 and subsequently, $(\Phi^{\circ}, \alpha^{\circ}) = (\Phi', \alpha') \circ (\Phi, \alpha)$ is a map of entailment systems. ■

Remark 15 In addition to the above result, it is not difficult to see that the composition of two conservative maps of entailment systems is itself conservative. Neither is it hard to see that the composition of maps entailment system is associative—it follows from the associativity of functor composition and vertical composition of natural transformations, respectively. □

These results regarding the composition of entailment system maps were the final brick needed in the construction of the category of entailment systems. It is now a simple matter to collect these bricks into the following trivial definition:

Definition 51 (The category of entailment systems) Let `Ent` denote the category of entailment systems. Its objects are all entailment systems as presented in Definition 42 and its morphisms are maps of entailment systems as presented in Definition 49. □

This concludes the treatment of the category of entailment systems, we will now look at the corresponding category for institutions.

4.6 The Category of Institutions

Recall from Section 4.1 that an institution contains structures representing both the syntax of a logic and its model-theoretic aspects. That is, not only does a *map of institutions* have to translate sentences and signatures, it is also required to translate models. Fortunately, the mapping of a logic's syntactic aspects has already been covered in the description of maps of entailment systems and as will be shown, it is possible to directly adopt these ideas in the institution case.

Informally, the mapping of a logic's model-theoretic aspects must be such that the invariance of truth is guaranteed. Obviously, it is desirable to avoid maps which take a model with one semantics to a model with a completely different semantics! Note that this will be a map between two *Mod* functors, that is, a natural transformation is called for. Additionally, similar to how *Mod* maps models in reverse, our natural transformation will map the *Mod* functors in reverse. The underlying reason for this is the same as the previously discussed reason why the *Mod* functor maps models in the reverse direction, that is, we wish to avoid ending up in inconsistent situations. The natural transformation which will perform this task shall be denoted β.

First, the formal definition will be given then a few concrete examples will illustrate the mechanics of this map.

Definition 52 (Map of institutions) Let

$$\mathscr{I} = (\mathtt{Sign}, Sen, Mod, \models) \text{ and } \mathscr{I}' = (\mathtt{Sign}', Sen', Mod', \models')$$

be two institutions. Then a map from $\mathscr{I}$ to $\mathscr{I}'$ is a triple $(\Phi, \alpha, \beta) : \mathscr{I} \longrightarrow \mathscr{I}'$ where

$$\mathtt{Th}_0 \xrightarrow{\Phi} \mathtt{Th}_0'$$

is an α-sensible functor from the category of $\mathscr{I}$-theories to the category of $\mathscr{I}'$-theories, and the natural transformations

$$Sen \xrightarrow{\alpha} Sen' \circ \mathcal{S}' \circ \Phi \circ \mathcal{T} \quad \text{and} \quad Mod' \circ \mathcal{S}' \circ \Phi \circ \mathcal{T} \xrightarrow{\beta} Mod$$

are such that the condition

$$M' \models'_{\Sigma'} \alpha_\Sigma(\varphi) \iff \beta_\Sigma(M') \models_\Sigma \varphi \tag{4.29}$$

hold for each $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$, $\varphi \in Sen(\Sigma)$, and $M' \in Mod'(\Sigma')$ with $\Sigma' = \mathcal{S}'\Phi\mathcal{T}(\Sigma)$. □

Remark 16 Recall that an α-sensible functor is such that conditions 49(i) and 49(ii) are upheld. In other words, these conditions are implicitly required for the map of institutions as well. □

Remark 17 Note that the Φ functor in the above definition actually does more work than is required. It is easy to see that all which is really needed for the map of institutions is a functor between the respective signature category of the institutions. However, by using a functor between the categories of theories instead, the extension to a map of logics will be very natural. This will become clear in Section 4.7. □

As a first concrete example of this type of mapping, we may consider the institutions $\mathscr{I}^{Eq}$ and $\mathscr{I}^{Fo}$ as presented in Section 4.1.1 and Section 4.1.2, respectively.

Example 37 We have in Example 35 already constructed the map between the entailment systems $\mathscr{E}^{Eq}$ and $\mathscr{E}^{Fo}$. Let Φ and α be as defined in that example. It is now interesting to see whether—given an appropriate β—the map $(\Phi, \alpha, \beta) : \mathscr{I}^{Eq} \longrightarrow \mathscr{I}^{Fo}$ is a map of institutions. This is definitely the case and since we in Definition 39 fixed Mod^{Fo} to be precisely the same as Mod^{Eq}, that is, β is nothing more than the identity natural transformation!

As a concrete example, let A be the classic algebra for the signature $\Sigma_{NATBOOL}$ having X as its associated family of variables. It is then easy to see using the equational and first order satisfaction relations—given in Definition 34 and 40—that for the equation $(\forall X)\; x = x * Succ(0)$, $x \in X_{nat}$, and first order formula

$$\alpha_{\Sigma_{NATBOOL}}((\forall X)\; x = x * Succ(0)) = (\forall x : nat)\; x{=}x * Succ(0),$$

we have, just as expected,

$$A \models^{Eq} (\forall X)\; x = x * Succ(0) \iff A \models^{Fo} (\forall x : nat)\; x = x * Succ(0).$$

□

Let us consider a more interesting mapping, namely the one from the many-sorted equational institution, $\mathscr{I}^{Eq}$, to its unsorted variant. This type of map is by its nature lossy, that is, some of the structure of the many-sorted institution will be lost. However, that it is at all possible to define this type of map is in of itself interesting—and sometimes quite useful as will be briefly discussed in Chapter 6. Only a short informal description of the unsorted equational institution will be given as its structure is quite simple and its use will become obvious from the example that follow. First, let

$$\mathscr{I}^{1Eq} = (\mathtt{Sign}^{1Eq}, Sen^{1Eq}, Mod^{1Eq}, \models^{1Eq})$$

be this institution. Then a signature $\Pi \in \mathrm{Ob}(\mathtt{Sign}^{1Eq})$ is a family of operations indexed by a natural number which indicates the operation arity, it may of course have an associated *set* of variables—see Remark 8. The sentences are equations of unsorted and unquantified terms, that is, the set of variables does not have to be mentioned in the equation. The models are unsorted algebras where an algebra $A \in \mathrm{Ob}(Mod^{1Eq}(\Pi))$ consists of a single carrier set, A, and for each operation $\omega \in (\Pi_n)_{n\in\mathbb{N}}$, a function $A_\omega : A^n \longrightarrow A$, where A^n is making use of the familiar notation where

$$A^n = \underbrace{A \times A \times \ldots \times A}_{\text{n times}}.$$

Finally, using the obvious inductive definition of the evaluation function $A(\mathfrak{v})$, the satisfaction relation is such that

$$A \models^{1Eq}_{\Pi} t = t' \iff A(\mathfrak{v})(t) = A(\mathfrak{v})(t')$$

hold for each unsorted signature Π, equation $(t = t') \in Sen^{1Eq}(\Pi)$, unsorted algebra $A \in \mathrm{Ob}(Mod^{1Eq}(\Pi)$, and variable assignment $\mathfrak{v} : X \longrightarrow A$.

Example 38 Put shortly, the map to describe is $(\Phi, \alpha, \beta) : \mathscr{I}^{Eq} \longrightarrow \mathscr{I}^{1Eq}$. Let $\mathtt{Th}_0^{Eq}$ denote the category of many-sorted equational theories and, similarly, let $\mathtt{Th}_0^{1Eq}$ denote the category of unsorted equational theories. Then the functor $\Phi : \mathtt{Th}_0^{Eq} \longrightarrow \mathtt{Th}_0^{1Eq}$ is such that

$$\Phi((S,\Omega),\Gamma) = (\Pi, \mathcal{P}\alpha_{(S,\Omega)}(\Gamma)),$$

where an operation $\omega \in \Pi_n$ if $(\omega, s_1, \ldots, s_n, s) \in \Omega$ for some $s, s_1, \ldots, s_n \in S$ and $n \in \mathbb{N}$, in other words, the sorts which are operated upon are forgotten and only the operation name and rank is retained. The natural transformation $\alpha : Sen^{Eq} \longrightarrow Sen^{1Eq}\mathcal{S}^{1Eq}\Phi\mathcal{T}^{Eq}$ is such that for each many-sorted equation $(\forall X)\, t = t'$,

$$\alpha_{(S,\Omega)}((\forall X)\, t = t') = (u = u')$$

where u, u' are the same as t, t' but with operations replaced by their unsorted counterparts. Finally, the natural transformation $\beta : Mod^{1Eq}\mathcal{S}^{1Eq}\Phi\mathcal{T}^{Eq} \longrightarrow Mod^{Eq}$ is such that, given any many-sorted signature $\Sigma = (S,\Omega) \in \mathrm{Ob}(\mathtt{Sign}^{Eq})$ and corresponding unsorted signature $\Pi = \mathcal{S}^{1Eq}\Phi\mathcal{T}^{Eq}(\Sigma)$, there is for each Π-algebra A an associated many-sorted algebra $A' = \beta_\Sigma(A)$. This algebra has for each $s \in S$, the carrier set $A'(s) = A$. Additionally, for each function $A_\omega : A^n \longrightarrow A$ and operation $\omega : s_1 \times \ldots \times s_n \longrightarrow s \in \Omega$, $s, s_1, \ldots, s_n \in S$, $n \in \mathbb{N}$, the function $A'(\omega) : A'(s_1) \times \ldots \times A'(s_n) \longrightarrow A'(s)$ is such that $A'(\omega) = A_\omega$. From this construction it directly follows that

$$A \models^{1Eq}_{\Pi} \alpha_\Sigma(\varphi) \iff \beta_\Sigma(A) \models^{Eq}_{\Sigma} \varphi.$$

To make the example more concrete, consider the many-sorted signature

$$\Sigma = (\{nat, bool\}, \{0 :\longrightarrow nat, Succ : nat \longrightarrow nat, IsZero : nat \longrightarrow bool\})$$

and the many-sorted equation $\varphi = ((\forall X)\ x = Succ(Succ(x)))$. Finally, let the unsorted algebra A be such that $A = \{☺, ☹\}$, $A_0 = ☹$, $A_{IsZero}(x) = ☺$, and

$$A_{Succ}(x) = \begin{cases} ☺ & \text{if } x = ☹ \\ ☹ & \text{if } x = ☺. \end{cases}$$

Then, following the description above, we have $\Phi(\Sigma,\Gamma) = (\Pi, \mathcal{P}\alpha_\Sigma(\Gamma))$ for each set of axioms $\Gamma \in \mathcal{P}Sen^{Eq}(\Sigma)$ and where Π is such that $0 \in \Pi_0$, $Succ \in \Pi_1$, and $IsZero \in \Pi_1$. Further, $\alpha_\Sigma(\varphi) = (x = Succ(Succ(x)))$ and $A' = \beta_\Sigma(A)$ is such that

$$\begin{aligned} A'(nat) &= \{☺, ☹\} \\ A'(bool) &= \{☺, ☹\} \\ A'(0 :\longrightarrow nat) &= ☹ \\ A'(IsZero : nat \longrightarrow bool)(x) &= ☺ \\ A'(Succ : nat \longrightarrow nat)(x) &= \begin{cases} ☺ & \text{if } x = ☹ \\ ☹ & \text{if } x = ☺. \end{cases} \end{aligned}$$

Is it now the case that

$$A \models^{1Eq}_{\Pi} x = Succ(Succ(x)) \iff A' \models^{Eq}_{\Sigma} (\forall X)\ x = Succ(Succ(x)) \tag{4.30}$$

holds? Since there are only two possible assignments for the variable x, it is easy to check each possibility. If $\mathfrak{v} : X \longrightarrow A$, $x \in X_{nat}$, and $\mathfrak{v}(x) = ☺$, then

$$A(\mathfrak{v})(x) = ☺ = Succ(☹) = Succ(Succ(☺)) = A(\mathfrak{v})(Succ(Succ(x))).$$

Dually, if $\mathfrak{v}(x) = ☹$, then

$$A(\mathfrak{v})(x) = ☹ = Succ(☺) = Succ(Succ(☹)) = A(\mathfrak{v})(Succ(Succ(x))).$$

In other words, the left-hand side of the equivalence in Eq. (4.30) is true in each case. By performing a similar check of the right-hand side we will find that it also is true in both possible cases. Subsequently, we conclude that Eq. (4.30) holds. □

As an interesting alternative to the previous example, we may consider the reverse map. That is, the map from the unsorted equational logic to its many-sorted counterpart. To show the structure of this map, a brief description is provided in the following example.

Example 39 The map in question is $(\Phi, \alpha, \beta) : \mathscr{I}^{1Eq} \longrightarrow \mathscr{I}^{Eq}$ and it is simply such that $\Phi(\Pi,\Gamma) = (\Sigma, \mathcal{P}\alpha_\Pi(\Gamma))$ where, using a newly introduced sort s,

$$\Sigma = (\{s\}, \{\omega : s^n \longrightarrow s \mid \omega \in \Pi_n, n \in \mathbb{N}\}).$$

For some unsorted signature with associated set of variables X we let $\alpha_\Pi(t = t') = ((\forall Y)\ u = u')$ for each $(t = t') \in Sen^{1Eq}(\Pi)$ and where $Y_u = X$ and u, u' are identical to t, t' but with each unsorted operation exchanged to its many-sorted namesake. Finally, given some unsorted signature Π, the natural transformation β is such that for any many-sorted Σ-algebra A with $\Sigma = \mathcal{S}^{Eq}\Phi\mathcal{T}^{1Eq}(\Pi)$, $A' = \beta_\Pi(A)$ is an unsorted Π-algebra such that $A' = A(s)$ and having for each function $A(\omega) : A(s)^n \longrightarrow A(s)$, the function $A'_\omega : A'^n \longrightarrow A'$ defined by $A'_\omega = A(\omega)$. From this construction we then have

$$A \models^{Eq}_{\Sigma} \alpha_\Pi(\varphi) \iff \beta_\Pi(A) \models^{1Eq}_{\Pi} \varphi.$$

To conclude the example, we show that β is a natural transformation. That is, that the diagram

$$\begin{tikzcd}
\Pi' \arrow[d, "\mu"'] & Mod^{Eq}\mathcal{S}^{Eq}\Phi\mathcal{T}^{1Eq}\Pi \arrow[r, "\beta_\Pi"] \arrow[d, "Mod^{Eq}\mathcal{S}^{Eq}\Phi\mathcal{T}^{1Eq}\mu"'] & Mod^{1Eq}\Pi \arrow[d, "Mod^{1Eq}\mu"] \\
\Pi & Mod^{Eq}\mathcal{S}^{Eq}\Phi\mathcal{T}^{1Eq}\Pi' \arrow[r, "\beta_{\Pi'}"'] & Mod^{1Eq}\Pi'
\end{tikzcd}$$

commutes, or stated equivalently, that

$$Mod^{1Eq}\mu \circ \beta_\Pi = \beta_{\Pi'} \circ Mod^{Eq}\mathcal{S}^{Eq}\Phi\mathcal{T}^{1Eq}\mu \tag{4.31}$$

holds. Here $\mu : \Pi' \longrightarrow \Pi$ is a map of unsorted signatures and have the obvious semantics, that is, each operation $\omega \in \Pi'_n$, $n \in \mathbb{N}$, is mapped to the operation $\mu(\omega) \in \Pi_n$. Let A be some many-sorted algebra constructed from Π as per the earlier description. Then expansion of the left-hand side of Eq. (4.31) yield

$$Mod^{1Eq}(\mu)(\beta_\Pi(A)) = Mod^{1Eq}(\mu)(A') = A''$$

where A' is the unsorted Π-algebra as computed by β_Π. Using the intuitive definition of $Mod^{1Eq}(\mu)$ we then have $A(s) = A' = A''$ and $A(\omega) = A'_\omega = A''_{\mu(\omega)}$ for each operation $\omega \in \Pi'$. Turning our attention to the right-hand side of Eq. (4.31) we find—using $\mu' = \mathcal{S}^{Eq}\Phi\mathcal{T}^{1Eq}\mu$—that

$$\beta_{\Pi'}(Mod^{Eq}(\mathcal{S}^{Eq}\Phi\mathcal{T}^{1Eq}\mu)(A)) = \beta_{\Pi'}(Mod^{Eq}(\mu')(A)) = \beta_{\Pi'}(B') = B''.$$

The intuitive definition of Φ gives a μ' such that $\mu'_\Omega(\omega : s^n \longrightarrow s) = \mu(\omega) : s^n \longrightarrow s$ and $\mu'_S(s) = s$. Together with $\beta_{\Pi'}$, it gives $A(s) = B'(s) = B''$ and $A(\omega) = B'(\mu(\omega)) = B''_{\mu(\omega)}$, respectively. That is, $A'' = B''$ and we have therefore shown that Eq. (4.31) holds and by extension shown that β is a natural transformation. □

Recall Example 32 where the entailment system of an institution was presented. There, $\mathscr{I}^+$ denoted the entailment system constructed from the institution $\mathscr{I}$. It is also possible to expand this notion to establish a connection between maps of institutions and a map of their corresponding entailment systems. The following proposition details this concept. The proof will not be given here, instead the interested reader is referred to [31].

Proposition 10 *Let $(\Phi, \alpha, \beta) : \mathscr{I} \longrightarrow \mathscr{I}'$ be a map between the institutions $\mathscr{I}$ and $\mathscr{I}'$. Then $(\Phi, \alpha) : \mathscr{I}^+ \longrightarrow \mathscr{I}'^+$ is a map of the entailment systems constructed as per Example 32.* □

PROOF See [31] ■

Just as in the case of maps of entailment systems, we have to define how maps of institutions are composed. Other than the fact that β maps model functors in reverse, no surprises are introduced.

Definition 53 (Composition of maps of institutions) Let

$$(\Phi,\alpha,\beta) : \mathscr{I} \longrightarrow \mathscr{I}' \text{ and } (\Phi',\alpha',\beta') : \mathscr{I}' \longrightarrow \mathscr{I}''$$

be maps of institutions, then their composition

$$(\Phi',\alpha',\beta') \circ (\Phi,\alpha,\beta) : \mathscr{I} \longrightarrow \mathscr{I}''$$

is such that

$$(\Phi',\alpha',\beta') \circ (\Phi,\alpha,\beta) = (\Phi' \circ \Phi, \alpha'_{\mathcal{S}'\Phi\mathcal{T}} \circ \alpha, \beta \circ \beta'_{\mathcal{S}'\Phi\mathcal{T}}). \tag{4.32}$$

□

Proposition 11 *The composition of institutions as given in Definition 53 is itself a map of institutions.* □

Proof Let $(\Phi,\alpha,\beta) : \mathscr{I} \longrightarrow \mathscr{I}'$ and $(\Phi',\alpha',\beta') : \mathscr{I}' \longrightarrow \mathscr{I}''$ be maps of institutions and let $(\Phi^\circ,\alpha^\circ,\beta^\circ)$ denote the composition $(\Phi',\alpha',\beta') \circ (\Phi,\alpha,\beta) : \mathscr{I} \longrightarrow \mathscr{I}''$, that is, $\Phi^\circ = \Phi' \circ \Phi$, $\alpha^\circ = \alpha'_{\mathcal{S}'\Phi\mathcal{T}} \circ \alpha$, and $\beta^\circ = \beta \circ \beta'_{\mathcal{S}'\Phi\mathcal{T}}$.

That the composed functor Φ° is α°-sensible is shown much like in Proposition 9 so that part will be omitted. Instead, we will concentrate on proving that the composed map fulfill the condition given in Eq. (4.29), that is, we wish to show that

$$M'' \models_{\Sigma''} \alpha^\circ_\Sigma(\varphi) \iff \beta^\circ_\Sigma(M'') \models_\Sigma \varphi$$

holds for each $M'' \in Mod''(\Sigma'')$, $\Sigma \in \mathrm{Ob}(\mathtt{Sign})$, and $\varphi \in Sen(\Sigma)$ with $\Sigma' = \mathcal{S}'\Phi\mathcal{T}(\Sigma)$ and $\Sigma'' = \mathcal{S}''\Phi'\mathcal{T}'(\Sigma')$. To show this is easy since we know that (Φ,α,β) and (Φ',α',β') are maps of institutions. Thus, the derivation

$$\begin{aligned} M'' \models''_{\Sigma''} \alpha^\circ_\Sigma(\varphi) &\iff M'' \models''_{\Sigma''} \alpha_{\Sigma'}(\alpha_\Sigma(\varphi)) \\ &\iff \beta'_{\Sigma'}(M'') \models'_{\Sigma'} \alpha_\Sigma(\varphi) \\ &\iff \beta_\Sigma(\beta'_{\Sigma'}(M'')) \models_\Sigma \varphi \\ &\iff \beta^\circ_\Sigma(M'') \models_\Sigma \varphi \end{aligned}$$

establishes the validity of Proposition 11. ■

Just as in the composition of maps of entailment systems, associativity follows directly from the properties of functors and natural transformations. All necessary groundwork has therefore been performed and it is now possible to define the category of institutions in a simple manner.

Definition 54 (The category of institutions) Let `Inst` denote the category of institutions. Its objects are all institutions as presented in Definition 29 and its morphisms are maps of institutions as presented in Definition 52. □

As is remarked in [31], this is not the only possible way to define this category. In e.g. [17], an alternative—but very similar—construction for institution morphisms is presented. The one given here follows Meseguers description and as he states: “In this presentation, I have favored the notion of a map of institutions because it fits well many natural examples and permits flexible ways of mapping theories.”

4.7 The Category of Logics

Very similar to how a logic was the combination of an institution and an entailment system, a map of logics is a combination of a map of institutions and a map of entailment systems. This may be formally described in the following, simple, way.

Definition 55 (Map of logics) Let

$$\mathscr{L} = (\mathtt{Sign}, Sen, Mod, \vdash, \models) \text{ and } \mathscr{L}' = (\mathtt{Sign}', Sen', Mod', \vdash', \models')$$

be two logics, then a *map of logics* is a triple $(\Phi, \alpha, \beta) : \mathscr{L} \longrightarrow \mathscr{L}'$ such that

$$(\Phi, \alpha, \beta) : (\mathtt{Sign}, Sen, Mod, \models) \longrightarrow (\mathtt{Sign}', Sen', Mod', \models')$$

is a map of the underlying institutions, and

$$(\Phi, \alpha) : (\mathtt{Sign}, Sen, \vdash) \longrightarrow (\mathtt{Sign}', Sen', \vdash')$$

is a map of the underlying entailment systems. □

We have already defined a map of logics throughout the two previous sections. It is therefore a simple matter of combining these into a single example.

Example 40 Let $\mathscr{L}^{Eq}$ and $\mathscr{L}^{Fo}$ be the logics of many-sorted equational logic and many-sorted first order logic, respectively. Then $(\Phi, \alpha, \beta) : \mathscr{L}^{Eq} \longrightarrow \mathscr{L}^{Fo}$ is a map between these logics where Φ, α, and β are as given in Example 35 and Example 37. □

As a further example, the interested reader may wish to confirm that the institution maps of Example 38 and 39 actually may be expanded to maps of logics. Finally, to conclude the section, we will of course define the category of logics.

Definition 56 (The category of logics) Let $\mathtt{Log}$ denote the category of logics. Its objects are all logics as presented in Definition 44 and it has as morphisms maps of logics as they were given in Definition 55. □

4.8 Proof Calculi and Logical Systems

As previously mentioned, we describe the proof structure, or proof calculi, separately from syntactic implication. The reasons for this abstraction will be clarified in this section and from it, a pleasantly general definition of proof calculi will be derived. Making a long story short, the underlying reason for not grafting a proof calculi onto the entailment relation directly is that doing so would necessarily bind us to that particular proof calculi. This is a limitation since there may exist several different proof calculi which yield the same logic. Further, these proof calculi may have properties which make a calculi ideal in one situation but less suited for some other situation. A logic well known for having multiple calculi is first order logic, where natural deduction, several other so-called sequent styled calculi, and the Hilbert styled calculi give precisely the same logic but whose proof process differ in crucial details. And these are only a few examples, several other proof calculi are possible for first order logic, see e.g. [33].

The advantage of separating entailment and proof calculi is therefore evident; we are able to choose the proof process most advantageous for our particular purpose.

Once a proof calculi has been chosen and combined with a logic, the resulting structure is known as a *logical system.*

The formal definition which follows is rather abstract, the strength of this is that it allows the encoding of a wide range of different notions of proof calculi. The negative consequence is that it becomes harder to grasp the underlying concept which is not as complex as it might seem. To help make these underlying concepts more apparent, two detailed examples of proof calculi will be given, one for equational and one for first order logic. In the first order case, the natural deduction calculi mentioned above will be described.

Definition 57 (Proof calculi) A proof calculus is a sixtuple

$$\mathscr{P} = (\mathtt{Sign}, Sen, \vdash, P, Pr, \pi)$$

such that

(i) $(\mathtt{Sign}, Sen, \vdash)$ is an entailment system;

(ii) $P : \mathtt{Th}_0 \longrightarrow \mathtt{Struct}$ is a functor taking theories to *proof-theoretic structures*;

(iii) $Pr : \mathtt{Struct} \longrightarrow \mathtt{Set}$ is a functor such that for a theory $T \in \mathrm{Ob}(\mathtt{Th}_0)$, the set $PrPT$ is the *set of proofs for T*; and

(iv) $\pi : Pr \circ P \longrightarrow Sen \circ \mathcal{S}$ is a natural transformation such that the image of the morphism $\pi_T : PrPT \longrightarrow Sen\mathcal{S}T$, for some theory $T = (\Sigma, \Gamma)$, is the set $\Gamma^{\bullet}$. □

Remark 18 As the reader with no doubt noticed, a key element for why the formal definition of proof calculi is so abstract is the introduction of yet another category, `Struct`, which is the category of proof-theoretic structures. It is well worth mentioning that the lack of detail about the actual structure of this category is not a mistake. In fact, this is the key to the abstract power of this categorized notion of proof calculi! □

As previously mentioned, when a logic is given a specific proof calculus the combined result is known as a logical system. The definition is straight-forward.

Definition 58 (Logical systems) A logical system, $\mathscr{S}$, is an 8-tuple

$$\mathscr{S} = (\mathtt{Sign}, Sen, Mod, \vdash, \models, P, Pr, \pi)$$

where

(i) $(\mathtt{Sign}, Sen, Mod, \vdash, \models)$ is a logic and

(ii) $(\mathtt{Sign}, Sen, \vdash, P, Pr, \pi)$ is a proof calculus. □

While a proof calculus by itself is not directly suited for efficient implementation it does, however, provide the basis for more efficient representations. In particular the notion of proof calculi may be extended to a structure known as *effective proof subcalculi*, this structure is described and used in [31] as a foundation in the formalization of logic programming. Effective proof subcalculi will not be covered in this thesis but in short it adds to a logic certain restrictions on for example the axioms the logic may possess and the conclusions one may prove from these axioms. Of course, these limitations should fulfill certain requirements in order to ensure that the power of the calculi is not compromised to the point where its usefulness vanishes.

Closing the topic of proof calculi and logical systems, two detailed examples of concrete proof calculi will be given. As before, the logics chosen for the examples are the familiar many-sorted equational and first order logics. For equational logic, a proof calculi will be constructed on the basis of so-called equational deduction. Similarly, for first order logic, natural deduction will form the basis of its notion of proof calculi given. For both equational and natural deduction, a brief general description will be given in order to recall the basic notions required. As mentioned before—but well worth repeating—these two notions of calculi are not the only ones possible for their respective logic. Other possibilities exist, each with their strengths and weaknesses.

Before tackling the examples, we will first informally recall the notion of *inference rules* as well as how they are applied—for a more thorough treatment of these notions please see e.g. [33]. An inference rule for some signature Σ and sentence functor *Sen* is simply a subset of $(Sen(\Sigma))^n \times Sen(\Sigma)$ where $n \in \mathbb{N}$. Consider for example the inference rule

$$\{((\varphi_1, \ldots, \varphi_n), \varphi) \mid \text{conditions}\}$$

or as it is typically written;

$$\frac{\varphi_1 \quad \ldots \quad \varphi_n}{\varphi} \quad \text{conditions.} \tag{4.33}$$

Informally, the inference rule of Eq. (4.33) states that, given a theory $T = (\Sigma, \Gamma)$, if the entailment relation is such that $\Gamma \vdash_\Sigma \varphi_1, \ldots, \Gamma \vdash_\Sigma \varphi_n$ then we may—provided the given conditions are satisfied—draw the conclusion that also $\Gamma \vdash_\Sigma \varphi$ hold. In the special case where $n = 0$ we simply have the inference rule $((), \varphi)$ for some $\varphi \in Sen(\Sigma)$. Of course, this rule may also be written

$$\frac{}{\varphi} \quad \text{conditions} \tag{4.34}$$

in the commonly used notation. Note that for an empty inference rule—such as in Eq. (4.34)—the sentence φ will always hold and in essence, this causes φ to become a concrete axiom in the logic, one which is specific for the particular proof calculi to which it belongs.

With inference rules added to our bag of tools, we may proceed to the examples. The following subsection contains the example covering equational logic and the subsection that follows it will describe a proof calculi for first order logic.

4.8.1 Proof Calculi for Equational Logic

We are now able to describe one possible proof calculus for equational logic. As mentioned, this calculus will be based on so-called equational deduction which here will be somewhat extended in order to manage the added difficulty of many-sorted signatures. For a thorough introduction to these notions see e.g. [23]. First we need to establish what equational deduction is: Equational deduction is a rather uncomplicated means of determining syntactic implication using five inference rules. These are, for each equational signature $\Sigma = (S, \Omega)$ with associated families of variables X and Y, as follows:

$$\frac{}{(\forall X)\, t = t} \quad t \in T_\Sigma X \tag{4.35}$$

which informally states that a term is always equal to itself, in other words, equality is reflexive—note that this rule will also act as an axiom in the calculus;

$$\frac{(\forall X)\ t = u}{(\forall X)\ u = t} \quad t,u \in T_\Sigma X, \tag{4.36}$$

that is, equality is symmetric;

$$\frac{(\forall X)\ t = u,\ (\forall X)\ u = v}{(\forall X)\ t = v} \quad t,u,v \in T_\Sigma X, \tag{4.37}$$

that is, equality is transitive;

$$\frac{(\forall X)\ t = u}{(\forall Y)\ t\sigma = u\sigma} \quad t,u \in T_\Sigma X, \sigma : X \longrightarrow T_\Sigma Y, \tag{4.38}$$

where σ is a substitution—we say that equality has the *substitution property*; and finally

$$\frac{(\forall X)\ t_1 = t_1', \ldots, (\forall X)\ t_n = t_n'}{(\forall X)\ u\sigma = u\tau} \quad \begin{array}{l} t_1,\ldots,t_n,t_1',\ldots,t_n' \in T_\Sigma X, \\ n \geq 1, u \in T_\Sigma Y, \text{ and} \\ \sigma,\tau : X \longrightarrow T_\Sigma Y \text{ where either} \\ \sigma(y) = \tau(y) \text{ or } \sigma(y) = t_i, \tau(y) = t_i' \\ \text{for all } y \in Y, 1 \leq i \leq n \end{array} \tag{4.39}$$

which states that equality uphold the so-called *congruence property*. These inference rules are fairly straight-forward—even Eq. (4.39) which informally states that substituting variables by equivalent terms is acceptable.

Using this notion of equational deduction, we shall now construct the sought after proof calculus which will be denoted

$$\mathscr{P}^{Eq} = (\mathtt{Sign}^{Eq}, Sen^{Eq}, \vdash^{Eq}, P^{Eq}, Pr^{Eq}, \pi^{Eq}).$$

In order for the definition to be general with regards to signature, we simply state that $(\mathtt{Sign}^{Eq}, Sen^{Eq}, \vdash^{Eq})$ represents *some* equational entailment system, that is, the specifics of the $\vdash^{Eq}$-relation is left undefined. It remains to show the appearance of P^{Eq}, Pr^{Eq}, and π^{Eq}. First, let $P^{Eq} : \mathtt{Th}_0 \longrightarrow \mathtt{Cat}$ be a functor taking a functor $T = (\Sigma, \Gamma)$ with associated family of Σ-variables X to the category $P^{Eq}(T)$ having

$$\mathrm{Ob}(P^{Eq}(T)) = T_\Sigma X$$

and for each $t, t' \in T_\Sigma X$,

$$\begin{aligned} \mathrm{Hom}_{P^{Eq}(T)}(t,t') = \{\varphi_1, \ldots, \varphi_n \mid n \geq 1 \text{ and } \varphi_1, \ldots, \varphi_n \text{ is a } \textit{derivation} \\ \textit{sequence} \text{ with } \varphi_n = ((\forall X)\ t = t')\}. \end{aligned} \tag{4.40}$$

By $\varphi_1, \ldots, \varphi_n$ being a derivation sequence is meant that for each φ_i, $1 \leq i \leq n$, it is the case that either $\varphi_i \in \Gamma$ or there exists an inference rule having as an element, $((\varphi_{j_1}, \ldots, \varphi_{j_m}), \varphi_i)$ where $j_1, \ldots, j_m < i$ and $m \geq 0$. The following brief example illustrates a concrete morphism in a category constructed by the P^{Eq} functor.

Example 41 Using $\Sigma_{\mathrm{NATLIST}}$ with associated family of variables X' and using as set of axioms, $\Gamma_{EqNatList}$ from Example 34, we may consider the equation

$$(\forall X')\ Empty(Tail(Cons(x', Nil))) = Empty(Nil).$$

This equation will in $P^{Eq}(T_{EqNatList})$ be represented by a morphism having the appearance

$$Empty(Tail(Cons(x',Nil))) \xrightarrow{\varphi_1;\varphi_2} Empty(Nil)$$

where

$$\varphi_1 = ((\forall X)\ Empty(Tail(Cons(x,y))) = Empty(y))$$

follows from the inference rule of Eq. (4.39) with $u = Empty(z)$ and the substitutions $\sigma(z) = Tail(Cons(x,y))$, $\tau(z) = y$, taking advantage of the axiom in Eq. (4.20); and where

$$\varphi_2 = ((\forall X')\ Empty(Tail(Cons(x',Nil))) = Empty(Nil))$$

follows from φ_1 using inference rule Eq. (4.38) with $\sigma : X \longrightarrow T_\Sigma X'$ such that $\sigma(x) = x'$ and $\sigma(y) = Nil$. □

It is easy to convince oneself that morphism composition in this category is nothing more than concatenation of the derivation sequences. Note that the inference rules of Eq. (4.35), Eq. (4.36), and Eq. (4.37) are represented directly in the structure of this category. That is, the reflexivity rule is represented by the identity morphisms. The symmetry rule states that all morphisms in $P^{Eq}(T)$, for some theory T, are isomorphic. Finally, the transitivity rule is directly represented by morphism composition.

Having specified the P^{Eq} functor, the $Pr^{Eq} : \mathtt{Cat} \longrightarrow \mathtt{Set}$ functor is given the uncomplicated definition where

$$Pr^{Eq}(P^{Eq}(T)) = \{(X,t,p,t') \mid p \in \mathrm{Hom}_{P^{Eq}(T)}(t,t'), t,t' \in T_\Sigma X\} \tag{4.41}$$

for each theory $T = (\Sigma,\Gamma)$ having associated family of variables X. Informally, this states that $Pr^{Eq}P^{Eq}T$ is the set of all equations provable from the theory T including their proof. Subsequently, $\pi^{Eq} : Pr^{Eq} \circ P^{Eq} \longrightarrow Sen^{Eq} \circ S$ simply projects the equation from these triples, that is,

$$\pi^{Eq}_T(X,t,p,t') = ((\forall X)\ t = t')$$

for each $(X,t,p,t') \in \mathrm{Ob}(Pr^{Eq}P^{Eq}T)$. In other words, π^{Eq}, forgets the proof and only remembers the proven equation. Of course, it is required to establish that π^{Eq} really is a natural transformation, this is done in the following proposition.

Proposition 12 *Let $\pi^{Eq} : Pr^{Eq}P^{Eq} \longrightarrow Sen^{Eq}S$ be as given above, then π^{Eq} is a natural transformation.* □

PROOF For simplicity, let in this proof $Sen = Sen^{Eq}$, $P = P^{Eq}$, $Pr = Pr^{Eq}$, and $\pi = \pi^{Eq}$. Then following the usual procedure, to show that π is a natural transformation, it is required to show that the diagram

$$\begin{tikzcd}
T \arrow[d, "\mu_{Th}"'] & PrPT \arrow[r, "\pi"] \arrow[d, "PrP\mu_{Th}"'] & Sen\mathcal{S}T \arrow[d, "Sen\mathcal{S}\mu_{Th}"] \\
U & PrPU \arrow[r, "\pi"'] & Sen\mathcal{S}U
\end{tikzcd}$$

commutes, that is, $Sen\mathcal{S}\mu_{Th} \circ \pi = \pi \circ PrP\mu_{Th}$. This will be done by showing that

$$(Sen\mathcal{S}\mu_{Th} \circ \pi_T)(x) = (\pi_U \circ PrP\mu_{Th})(x) \tag{4.42}$$

for some $x = (X,t,p,t') \in PrPT$ and theory $T = (\Sigma,\Gamma)$. The left-hand side of Eq. (4.42) is then

$$\begin{aligned}(Sen\mathcal{S}\mu_{Th} \circ \pi_T)(x) &= Sen\mathcal{S}\mu_{Th}(\pi_T(X,t,p,t')) \\ &= Sen\mathcal{S}\mu_{Th}((\forall X)\ t = t') \\ &= ((\forall \mu(X))\ \mu(t) = \mu(t'))\end{aligned}$$

while the right-hand side of Eq. (4.42) is

$$\begin{aligned}(\pi_U \circ PrP\mu_{Th})(x) &= \pi_U(PrP\mu_{Th}(X,t,p,t')) \\ &= \pi_U(\mu(X),\mu(t),p',\mu(t'))) \\ &= ((\forall \mu(X))\ \mu(t) = \mu(t'))\end{aligned}$$

Where p' is the proof of $(\forall \mu(X))\ \mu(t) = \mu(t')$ in the U theory. We know that there exists such a proof due to the construction of μ_{Th}, see Definition 46. Clearly, both the left-hand side and the right-hand side yield the same equation and we may subsequently conclude that π is a natural transformation. ■

The proof process which P^{Eq} and Pr^{Eq} perform is both sound and complete—see e.g. [27] for formal proof—and as a result, it is clear from its construction that the image of π_T is the set of equations provable from the theory $T = (\Sigma,\Gamma)$. In other words, the image contains exactly the sentences of $\Gamma^\bullet$. Just as required in Definition 57(iv).

This rather theoretical description may be more easily understood through the help of a complete example.

Example 42 In the chapter introduction, the reason for thoroughly examining these issues was motivated, in particular we wondered about the following question:

Is it **always** true that $Head(Tail(Cons(x,Cons(y,Nil)))) = Head(Cons(y,z))$?

It is now possible to not only pose this question in a formal way but also conclusively answer it. Using the equational theory given in Example 34 we are in essence asking whether

$$\begin{aligned}&\Gamma_{\text{EqNatList}} \vdash (\forall \{\{x,y\}_{nat}, \{z\}_{natlist}\}) \\ &\qquad Head(Tail(Cons(x,Cons(y,Nil)))) = Head(Cons(y,z)).\end{aligned}$$

holds. This, in turn, asks whether there exists a derivation sequence $p = \varphi_1,\ldots,\varphi_n$ such that

$$\begin{aligned}&[\{\{x,y\}_{nat}, \{z\}_{natlist}\}, Head(Tail(Cons(x,Cons(y,Nil)))), \\ &\qquad\qquad p, Head(Cons(y,z)] \in PrP(T_{EqNatList}).\end{aligned} \tag{4.43}$$

We may easily construct one possible p by letting $p = \varphi_1, \varphi_2, \ldots, \varphi_9$ such that

$$\begin{aligned}
&\varphi_1 = (\forall X)\ Head(l_1) = Head(l_1)\\
&\varphi_2 = (\forall X)\ Head(Tail(Const(e_1, l_2))) = Head(l_2)\\
&\varphi_3 = (\forall X)\ Head(Tail(Const(e_2, Cons(e_3, l_3)))) = Head(Cons(e_3, l_3))\\
&\varphi_4 = (\forall X)\ Head(Cons(e_3, Nil)) = e_3\\
&\varphi_5 = (\forall X)\ Head(Cons(e_3, l_4)) = e_3\\
&\varphi_6 = (\forall X)\ e_3 = Head(Cons(e_3, l_4))\\
&\varphi_7 = (\forall X)\ Head(Cons(e_3, Nil)) = Head(Cons(e_3, l_4))\\
&\varphi_8 = (\forall X)\ Head(Tail(Const(e_2, Cons(e_3, l_3)))) = Head(Cons(e_3, l_4))\\
&\varphi_9 = (\forall \{\{x, y\}_{nat}, \{z\}_{natlist}\})\\
&\qquad Head(Tail(Const(x, Cons(y, Nil)))) = Head(Cons(y, z)).
\end{aligned}$$

for some temporary variables $e_1, \ldots, e_3 \in X_{nat}$ and $l_1, \ldots, l_4 \in X_{natlist}$. The exact details behind the above derivation steps will not be accounted for, this is left as an exercise for the reader. Instead we observe that these together describe the morphism

$$Head(Tail(Const(x, Cons(y, Nil)))) \xrightarrow{p} Head(Cons(y, x))$$

which we may find in $P^{Eq}(T_{EqNatList})$. This morphism will subsequently be extracted by Pr^{Eq} so Eq. (4.43) holds and we may therefore conclude that the question posed has an affirmative answer. □

We will revisit this proof calculi for equational logic in Section 4.9 where the category of proof calculi is described. In that section an example will be given where we go from the equational calculi to the first order calculi which will be presented next.

4.8.2 Proof Calculi for First Order Logic

Moving on to the first order case we first recall that the basis for the first order logic proof calculi presented here is natural deduction. This form of deduction is well known and has a rather simple structure, although it is somewhat more complex than equational deduction. For this reason, a thorough description of natural deduction will be omitted and only a brief overview will be given. For a more formal treatment of natural deduction in the many-sorted case see e.g. [5, 37].

Natural deduction which was independently invented by S. Jaskowsky and G. Gentzen in the mid 1930's is a calculi intended to match human reasoning [34]. One of the most noticeable differences from other calculi is the possibility of doing *subderivations* (or *subproofs*) which in essence means that it is possible to make arbitrary *assumptions* during the proof process. Consider a brief example; Assume that the formula φ holds, then if we, given this assumption, are able to deduce the truth of both φ' and $\neg\varphi'$ then it is possible to draw the conclusion that $\neg\varphi$ must hold. This particular example is one of the inference rules of natural deduction. Unfortunately, it is not possible to directly express rules of this form in the previously defined notion of inference rules—we need to generalize it slightly. Recall that, given a theory $T = (\Sigma, \Gamma)$, an inference rule

$$\frac{\varphi_1 \quad \ldots \quad \varphi_n}{\varphi} \quad \text{conditions}$$

means that under the given conditions, if $\Gamma \vdash_\Sigma \varphi_1, \ldots, \Gamma \vdash_\Sigma \varphi_n$ then $\Gamma \vdash_\Sigma \varphi$. We may therefore equivalently write

$$\frac{\Gamma \vdash_\Sigma \varphi_1 \quad \ldots \quad \Gamma \vdash_\Sigma \varphi_n}{\Gamma \vdash_\Sigma \varphi} \quad \text{conditions.}$$

Using this notation, the inference rule given as a small example may then be written

$$\frac{\Gamma \cup \{\varphi\} \vdash^{Fo}_{\Sigma} \varphi' \quad \Gamma \cup \{\varphi\} \vdash^{Fo}_{\Sigma} \neg\varphi'}{\Gamma \vdash^{Fo}_{\Sigma} \neg\varphi}.$$

Using this new notation when needed, we now turn to a short description of the inference rules which lie at the heart of natural deduction. First we observe that most of them may be classified as either *introducing* or *eliminating* rules. Stated informally, given one or more formulas an introducing rules create a new formula. Dually, given one or more formulas, an eliminating rule extract a formula which is a component of one of the input formulas. Given a theory $T = (\Sigma, \Gamma)$ where $\Sigma = (S, \Omega)$ with associated family of variables X, the inference rules of [45]—somewhat simplified and adapted to our many-sorted logic—are

$$\frac{\Gamma \cup \{\varphi\} \vdash^{Fo}}{\Gamma \cup \{\varphi\} \vdash^{Fo} \varphi} \quad (4.44) \qquad \frac{\Gamma \vdash^{Fo} \varphi}{\Gamma \cup \Gamma' \vdash^{Fo} \varphi} \quad (4.45)$$

that is, an axiom may always be derived and adding further axioms never invalidate an already derived formula. Additionally, the introducing and eliminating rules are first

$$\begin{array}{cccc} \underline{\text{Introducing rule}} & & \underline{\text{Eliminating rule}} & \\ \dfrac{}{t = t} & (4.46) & \dfrac{\varphi \quad t = t'}{\varphi'} & (4.47) \end{array}$$

where $t, t' \in T_{\Sigma,s}X$, $s \in S$, and φ' is a formula identical to φ but where an arbitrary number of occurrences of the term t have been replaced by the term t'. The rest of the rules are more straight-forward—except the last one—and these are such that

$$\frac{\Gamma \cup \{\varphi\} \vdash^{Fo} \varphi' \quad \Gamma \cup \{\varphi\} \vdash^{Fo} \neg\varphi'}{\Gamma \vdash^{Fo} \neg\varphi} \quad (4.48) \qquad \frac{\neg\neg\varphi}{\varphi} \quad (4.49)$$

$$\frac{\varphi \quad \varphi'}{\varphi \wedge \varphi'} \quad (4.50) \qquad \frac{\varphi \wedge \varphi'}{\varphi}, \frac{\varphi \wedge \varphi'}{\varphi'} \quad (4.51)$$

$$\frac{\Gamma \cup \{\varphi\} \vdash^{Fo} \varphi'}{\Gamma \vdash^{Fo} \varphi \longrightarrow \varphi'} \quad (4.52) \qquad \frac{\varphi \quad \varphi \longrightarrow \varphi'}{\varphi'} \quad (4.53)$$

$$\frac{\varphi}{(\forall x : s)\ \varphi} \quad x \notin \text{free}(\varphi) \quad (4.54) \qquad \frac{(\forall x : s)\ \varphi}{\varphi''} \quad (4.55)$$

where $\varphi, \varphi', \varphi'' \in Sen^{Fo}(\Sigma)$, $s \in S$ and the φ'' of Eq. (4.55) is precisely the same as φ but having *each* occurrence of the variable x replaced by some arbitrary term $t \in T_{\Sigma,s}X$. As mentioned, these rules are relatively straight-forward and given the previously covered material, should not need extensive explanations. It should be mentioned that the inference rules of natural deduction may be presented in many different manners. Additionally, the exact number of rules may vary somewhat depending on how the underlying logic was defined.

The proofs of natural deduction are often thought of in terms of as so-called *proof trees*; the name stems from the appearance of the proofs with the proven formula as root and stacked inference rules forming the structure of the tree branches and leaves. No formal description of these proof trees will be given, they will instead be introduced in the form of a simple example.

Example 43 Suppose we wish to prove the first order sentence

$$(x \boldsymbol{\wedge} False) \longrightarrow (\forall y : bool)\ (True \boldsymbol{\wedge} y) = y). \tag{4.56}$$

using the theory T_{FoBool} given in Example 33. By taking advantage of the inference rules given above, we may construct a proof tree which shows the correctness of Eq. (4.56). So as to avoid confusion, please recall the difference between $\wedge$ and $\boldsymbol{\wedge}$; the former is part of the syntax introduced by the theory and the latter is part of the logic's so-called *metalanguage*.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{x=x}}{\Gamma_{FoBool} \cup \{x \wedge False\} \vdash^{Fo} x=x} \qquad \cfrac{\cfrac{\cfrac{}{v=v}}{\Gamma_{FoBool} \vdash^{Fo} v=v}}{\Gamma_{FoBool} \cup \{x \wedge False\} \vdash^{Fo} (\forall y : bool)\ y=y}}{\Gamma_{FoBool} \cup \{x \wedge False\} \vdash^{Fo} (x=x) \boldsymbol{\wedge} (\forall y : bool)\ y=y}}{(x \wedge False) \longrightarrow ((x=x) \boldsymbol{\wedge} ((\forall y : bool)\ y=y))}$$

The proof tree above—rooted at the bottom and having two leaves—shows that it is possible to derive Eq. (4.56) from the axioms of T_{FoBool}.

Note that we are forced to introduce a new, unique, variable when applying the introducing rule for $\forall$, that is, the rule of Eq. (4.54). If addition of a new variable is omitted, there is a risk that the proof becomes inconsistent. Consider for example the simple formula $x = True \longrightarrow (\forall x : s)\ x = True$. □

Of course, the proof calculi which will be presented here will, as part of its proof-theoretic structures, take advantage of proof trees. As a first step towards this description, there is a need to find an appropriate representation of these proof-theoretic structures and in this case, a category of what in [31] are termed *multicategories* will work very well. These multicategories are also called *strict monoidal categories* and are under that name covered in detail in [28]. Only a semi-formal definition will be given here—instead, the interested reader is referred to the above sources. The basic concept of a multicategory is that its objects are strings of *basic objects*, drawn from an alphabet defined separately from the category itself. The monoidal part of "free monoidal category" arise from it having an associated multiplication functor and an identity object. This is analogous to traditional monoids which commonly are defined as a pair $(S, \cdot)$ where S is a set and $\cdot$ is a binary relation with an identity element $i \in S$ such that for any $s \in S$, we have $i \cdot s = s \cdot i = s$, see e.g. [2]. The description of multicategory is made more formal by the following definition:

Definition 59 (Multicategories) A multicategory is a triple $\mathcal{M} = (\mathcal{O}, \mathtt{C}, \ddagger)$ where

(i) $\mathcal{O}$ is a set of *basic objects*, and

(ii) $\mathtt{C}$ is a category having as objects finite strings of basic objects, that is,

$$\mathrm{Ob}(\mathtt{C}) = \{A_1 \ddagger A_2 \ddagger \ldots \ddagger A_n \mid A_1, A_2, \ldots, A_n \in \mathcal{O}, n \geq 0\}.$$

The empty string, that is, when $n = 0$, is denoted $\varnothing$.

(iii) $\ddagger : \mathtt{C} \times \mathtt{C} \longrightarrow \mathtt{C}$ is a functor—typically expressed in an infix way—describing multiplication in such a way that for strings

$$\Gamma = A_1 \ddagger \ldots \ddagger A_n, \Delta = B_1 \ddagger \ldots \ddagger B_m \in \mathrm{Ob}(\mathtt{C}),$$

we have

$$\Gamma \ddagger \Delta = A_1 \ddagger \ldots \ddagger A_n \ddagger B_1 \ddagger \ldots \ddagger B_m \in \mathrm{Ob}(\mathtt{C}),$$

that is, the strings are concatenated. Similarly, the multiplication of $\mathtt{C}$-morphisms $\alpha : \Gamma \longrightarrow \Delta$ and $\beta : \Gamma' \longrightarrow \Delta'$ is another $\mathtt{C}$-morphism

$$\Gamma \ddagger \Gamma' \xrightarrow{\alpha \ddagger \beta} \Delta \ddagger \Delta'.$$ □

Remark 19 From this definition we clearly see the monoidal property as $\varnothing$ will act as the identity for the multiplication functor. That is, for any $\Gamma \in \mathrm{Ob}(\mathtt{C})$, we have $\Gamma \ddagger \varnothing = \varnothing \ddagger \Gamma = \Gamma$. □

Remark 20 From Definition 59(ii) it is obvious that for any multicategory $(\mathtt{C}, \mathcal{O}, \ddagger)$, it is the case that $\mathcal{O} \subseteq \mathrm{Ob}(\mathtt{C})$. □

The notion of multicategory may be extended to a category of multicategories, denoted `MultCat` whose objects are multicategories and morphisms are functors between them. These functors are defined in the natural manner subject to one unobvious restriction—basic objects must be mapped to basic objects. By this is meant that a morphism

$$(\mathtt{C}, \mathcal{O}, \ddagger) \xrightarrow{F} (\mathtt{C}', \mathcal{O}', \ddagger')$$

in `MultCat` is such that each basic object of $\mathcal{O}$ is mapped to a corresponding basic object in $\mathcal{O}'$. It is this `MultiCat` category which in this proof calculi shall take the place of the `Struct` category mentioned in Definition 57.

Having established what `Struct` is, it is possible to define the actual proof calculi for our notion of first order logic. This proof calculus will here be denoted

$$\mathscr{P}^{Fo} = (\mathtt{Sign}^{Fo}, Sen^{Fo}, \vdash^{Fo}, P^{Fo}, Pr^{Fo}, \pi^{Fo})$$

where $\mathtt{Sign}^{Fo}$, Sen^{Fo}, and $\vdash^{Fo}$ describe a first order entailment system, just as presented in Section 4.2. It remains to define P^{Fo}, Pr^{Fo}, and π^{Fo}.

We let $P^{Fo} : \mathtt{Th}_0 \longrightarrow \mathtt{MultiCat}$ be the functor such that, given some theory $T = (\Sigma, \Gamma) \in \mathrm{Ob}(\mathtt{Th}_0)$, $P^{Fo}(T)$ is the multicategory having $Sen^{Fo}\mathcal{S}(T)$ as set of basic objects, that is

$$\mathrm{Ob}(P^{Fo}(T)) = \{\varphi_1 \ddagger \ldots \ddagger \varphi_n \mid \varphi_1, \ldots, \varphi_n \in Sen^{Fo}\mathcal{S}(T), n \geq 0\}$$

and having as morphisms

$$\varphi_1 \ddagger \ldots \ddagger \varphi_n \xrightarrow{p} \varphi'_1 \ddagger \ldots \ddagger \varphi'_m \qquad n \geq 0, m \geq 1,$$

where $p = p_1 \ddagger \ldots \ddagger p_m$ is such that p_i, $1 \leq i \leq m$, is a proof tree of φ'_i which has as leafs sentences from no other sources than Γ and $\varphi_1, \ldots, \varphi_n$. For the special case when $n = 0$—that is, for a morphism of the form $p : \varnothing \longrightarrow \varphi'_1 \ddagger \ldots \ddagger \varphi'_m$—each sentence in the codomain is directly derivable from the axioms in the given theory. This is a crucial detail to keep in mind when we now turn to the definition of Pr^{Fo}.

We let $Pr^{Fo} : \texttt{MultiCat} \longrightarrow \texttt{Set}$ be a functor such that for each multicategory, say $(\mathcal{O}, \mathtt{C}, \ddagger)$, it is the case that

$$Pr^{Fo}(\mathcal{O}, \mathtt{C}, \ddagger) = \{\alpha \in \mathrm{Hom}_C(\varnothing, A) \mid A \in \mathcal{O}\},$$

which in the context of the previously defined proof-theoretic structures gives

$$Pr^{Fo}(P^{Fo}(T)) = \{\varnothing \xrightarrow{p} \varphi \mid \varphi \in Sen^{Fo}\mathcal{S}(T), p \in \mathrm{Hom}_{P^{Fo}(T)}(\varnothing, \varphi)\}.$$

The task of projecting the sentences from this is easy and we have for each proof $p : \varnothing \longrightarrow \varphi \in Pr^{Fo}P^{Fo}T$,

$$\pi^{Fo}(\varnothing \xrightarrow{p} \varphi) = \varphi.$$

Just as in the example showing the equational proof calculus, we must establish that π^{Fo} truly is a natural transformation.

Proposition 13 *Let $\pi^{Fo} : Pr^{Fo}P^{Fo} \longrightarrow Sen^{Fo}\mathcal{S}$ be as given above, then π^{Fo} is a natural transformation.* □

PROOF To avoid excessive notation, let in this proof $Sen = Sen^{Fo}$, $P = P^{Fo}$, $Pr = Pr^{Fo}$, and $\pi = \pi^{Fo}$. We then wish to show that

$$\begin{tikzcd}
T \arrow[d, "\mu_{Th}"'] & PrPT \arrow[r, "\pi_T"] \arrow[d, "PrP\mu_{Th}"'] & Sen\mathcal{S}T \arrow[d, "Sen\mathcal{S}\mu_{Th}"] \\
U & PrPU \arrow[r, "\pi_T"'] & Sen\mathcal{S}U
\end{tikzcd}$$

commutes, that is, $Sen\mathcal{S}\mu_{Th} \circ \pi = \pi \circ PrP\mu_{Th}$. This will be done by showing that

$$(Sen\mathcal{S}\mu_{Th} \circ \pi_T)(x) = (\pi_U \circ PrP\mu_{Th})(x) \tag{4.57}$$

for some $x = p : \varnothing \longrightarrow \varphi \in PrPT$ and theory $T = (\Sigma, \Gamma)$. The left-hand side of Eq. (4.57) is then

$$\begin{aligned}(Sen\mathcal{S}\mu_{Th} \circ \pi_T)(x) &= Sen\mathcal{S}\mu_{Th}(\pi_T(p : \varnothing \longrightarrow \varphi)) \\ &= Sen\mathcal{S}\mu_{Th}(\varphi)\end{aligned}$$

while the right-hand side of Eq. (4.57) is

$$\begin{aligned}(\pi_U \circ PrP\mu_{Th})(x) &= \pi_U(PrP\mu_{Th}(p : \varnothing \longrightarrow \varphi)) \\ &= \pi_U(p' : \varnothing \longrightarrow Sen\mathcal{S}\mu_{Th}(\varphi)) \\ &= Sen\mathcal{S}\mu_{Th}(\varphi).\end{aligned}$$

The remark which was made in the proof of Proposition 12 regarding p' still stands, its existence is guaranteed from the construction of μ_{Th}. Therefore, both the left-hand side and the right-hand side clearly yield the same result and we conclude that π is indeed a natural transformation. ■

It is clear from the construction of π^{Fo} that for some first order theory T, that the image of π^{Fo}_T is the set of sentences provable from the axioms of T. Proving soundness and completeness—in the form of the important Gödel's Completeness Theorem—of this proof process lies outside the scope of this thesis but the proof for the unsorted case may be looked up in e.g. [45].

This concludes the construction of the $\mathscr{P}^{Fo}$ proof calculus. We will now extend the notion of proof calculi to the world of categories.

4.9 The Categories of Proof Calculi and Logic Systems

The two major concepts left to describe are the extensions of proof calculi and logic systems into their respective category. As before, it is first necessary to establish precisely what constitutes a mapping in the context of proof calculi and logic systems. Fortunately, once a mapping of proof calculi has been described, the logic system mapping will follow without much effort.

Recall that a proof calculi consists in part of an entailment system, a map of proof calculi will subsequently consist in part of a map of entailment systems. Further, a map of proof calculi will also be required to translate the proofs of the source proof calculi into valid proofs in the target calculi. These notions are formalized in Definition 60 below.

Definition 60 (Map of proof calculi) Let

$$\mathscr{P} = (\mathtt{Sign}, Sen, \vdash, P, Pr, \pi) \text{ and}$$
$$\mathscr{P}' = (\mathtt{Sign}', Sen', \vdash', P', Pr', \pi')$$

be two proof calculi. Then a *map of proof calculi* $(\Phi, \alpha, \gamma) : \mathscr{P} \longrightarrow \mathscr{P}'$ is such that

$$(\mathtt{Sign}, Sen, \vdash) \xrightarrow{(\Phi,\alpha)} (\mathtt{Sign}', Sen', \vdash')$$

maps the embedded entailment system of $\mathscr{P}$ to the one of $\mathscr{P}'$, and

$$Pr \circ P \xrightarrow{\gamma} Pr' \circ P' \circ \Phi$$

is a natural transformation such that

$$\alpha_{\mathcal{S}} \circ \pi = \pi'_{\Phi} \circ \gamma. \tag{4.58}$$

□

Informally, one may say that Eq. (4.58) requires the map to be such that mapping the projected proofs of the source logic to sentences of the target logic yield the same result as mapping the proofs first and then projecting the sentences. This is mostly clearly demonstrated using an example. As previously mentioned, this example will consist of the construction of a map which takes the proof calculi of equational logic, $\mathscr{P}^{Eq}$, to the proof calculi of first order logic, $\mathscr{P}^{Fo}$ as they were given in Section 4.8.1 and Section 4.8.2, respectively.

Example 44 Let

$$\mathscr{P}^{Eq} = (\mathtt{Sign}^{Eq}, Sen^{Eq}, \vdash^{Eq}, P^{Eq}, Pr^{Eq}, \pi^{Eq})$$

denote the example equational proof calculus of Section 4.8.1 and, similarly, let

$$\mathscr{P}^{Fo} = (\mathtt{Sign}^{Fo}, Sen^{Fo}, \vdash^{Fo}, P^{Fo}, Pr^{Fo}, \pi^{Fo})$$

denote the first-order proof calculus of Section 4.8.2. We may construct the mapping

$$\mathscr{P}_{Eq} \xrightarrow{(\Phi,\alpha,\gamma)} \mathscr{P}_{Fo}$$

by letting Φ and α be as in Example 35, that is (Φ,α) is the map which takes the underlying entailment system of $\mathscr{P}^{Eq}$ to entailment system of $\mathscr{P}^{Fo}$. Turning to the construction of γ, we first observe that we are fortunate in so far that the inference rules of equational deduction are subsumed by those of natural deduction. In particular it is easy to see that the equational deduction inference rules of Eq. (4.35) through Eq. (4.39) may be represented using one or more applications of the Eq. (4.46) and Eq. (4.47) natural deduction inference rules. Using this observation, we may for each equational theory $T = (\Sigma, \Gamma)$, where $\Sigma \in \mathrm{Ob}(\mathtt{Sign}^{Eq})$ and $\Gamma \in \mathcal{P}Sen^{Eq}(\Sigma)$, construct the first-order proof

$$\gamma_T(X,t,p,t') = \varnothing \xrightarrow{p'} \alpha_\Sigma((\forall X)\ t = t') \tag{4.59}$$

where p' is a proof tree constructed from $p = \varphi_1, \ldots, \varphi_n$, $n \geq 1$, as per the following argument: We know from Eq. (4.40) that $\varphi_n = (\forall X)\ t = t'$, and that φ_i, $1 \leq i \leq n$, may be derived from $\varphi_1, \ldots, \varphi_{i-1}$ in a single equational deduction step. Further, since it is possible to "simulate" the inference rules of equational deduction using the natural deduction inference rules, we know that there exists proof trees p_1 and p'_1 such that $p' = p'_1 \circ p_1$ and

$$\varnothing \xrightarrow{p_1} \alpha_\Sigma(\varphi_1) \xrightarrow{p'_1} \alpha(\varphi_n).$$

Continuing, we know that there exists proof trees p_2 and p'_2 such that $p' = p'_2 \circ p_2 \circ p_1$ and

$$\varnothing \xrightarrow{p_1} \alpha_\Sigma(\varphi_1) \xrightarrow{p_2} \alpha_\Sigma(\varphi_1) \ddagger \alpha_\Sigma(\varphi_2) \xrightarrow{p'_2} \alpha(\varphi_n).$$

Inductively extending this argument will give that there exist proof trees $p_1, \ldots, p_n$ such that $p' = p_n \circ \ldots \circ p_1$ and

$$\varnothing \xrightarrow{p_1} \alpha_\Sigma(\varphi_1) \longrightarrow \ldots \longrightarrow \alpha_\Sigma(\varphi_1) \ddagger \ldots \ddagger \alpha_\Sigma(\varphi_{n-1}) \xrightarrow{p_n} \alpha(\varphi_n).$$

How to exactly find the proof trees $p_1, \ldots, p_n$ will here be left unspecified but it is worth noting that the search space will be finite since we may limit our search to rule instances having as output strings of length no greater than the length of φ_n and the maximum depth of each proof tree is bounded by the number of variables in the formula. While theoretically possible, such a brute force search is of course an unreasonable approach in practice. To make the situation more tractable, one could for example extend the notion of an equational deduction proof to include information of which particular rule instance was applied in each deductive step.

From the construction of this map it follows immediately that the condition of

Eq. (4.58) is upheld. More specifically, we have

$$\begin{aligned}(\alpha_{\mathcal{S}} \circ \pi^{Eq})_T(X,t,p,t') &= \alpha_{\mathcal{S}T}(\pi^{Eq}_T(X,t,p,t')) \\ &= \alpha_{\mathcal{S}T}((\forall X)\; t = t') \\ &= \pi_{\Phi T}(p' : \varnothing \longrightarrow \alpha_{\mathcal{S}T}((\forall X)\; t = t')) \\ &= \pi_{\Phi T}(\gamma_T(X,t,p,t')) \\ &= (\pi_\Phi \circ \gamma)_T(X,t,p,t').\end{aligned}$$

□

In light of the previous example, one might ask whether it is possible to construct a map when one is not fortunate enough to have a target proof calculi with a deductive system which subsumes the deductive system of the source proof calculi. The answer may be found by considering Eq. (4.58). Under the assumption that the source and target proof calculi have been fixed, we are able to alter only α and γ—the two natural transformations π and π' are part of the proof calculi. Therefore, since we are unable to translate proofs without loss of information, we will have construct α and γ such that this loss of information does not introduce inconsistencies. By constructing these two in a clever manner, it is indeed possible to construct maps from proof calculi of greater expressive power to less powerful ones. Put shortly, this situation is similar to the one presented in Example 38 where the institution of many-sorted equational logic was taken to its unsorted same. In both cases, some loss of information is unavoidable in the general case but this is sometimes an acceptable price to pay.

Composition of proof calculi is defined precisely in the manner one might expect. That is, the Φ and α components are composed in the style introduced in Definition 50 where composition of maps of entailment systems was given. Finally, the γ components are vertically composed—with care taken to not forget the stacked Φ component. The following more formal description illustrate this.

Definition 61 (Composition of maps of proof calculi) Let

$$\begin{tikzcd}\mathscr{P} \arrow[r, "(\Phi,\alpha,\gamma)"] & \mathscr{P}' \arrow[r, "(\Phi',\alpha',\gamma')"] & \mathscr{P}''\end{tikzcd}$$

be maps of proof calculi, then the composition of them,

$$\begin{tikzcd}\mathscr{P} \arrow[r, "(\Phi,\alpha,\gamma)\circ(\Phi',\alpha',\gamma')"] & \mathscr{P}''\end{tikzcd},$$

is such that

$$(\Phi,\alpha,\gamma)\circ(\Phi',\alpha',\gamma') = (\Phi'\circ\Phi, \alpha'_{\mathcal{S}'\Phi\mathcal{T}} \circ \alpha, \gamma'_\Phi \circ \gamma)$$

□

That composition of maps of proof calculi as given in Definition 61 yield a map of proof calculi and that composition is associative is shown much like in the previous maps so this will not be formulated in a proposition here. Instead, we simply conclude that—taking advantage of the results presented in this section—the category of proof calculi is defined as follows.

Definition 62 (The category of proof calculi) Let `PCalc` denote the category of proof calculi. Its objects are all proof calculi as described in Definition 57 and its morphisms are maps of proof calculi, given in Definition 60. □

Extending this definition to incorporate logic systems give us another category; the one for logical systems.

Definition 63 (The category of logical systems) Let `LogSys` denote the category of logical systems having as objects all logical systems as per Definition 58 and having as morphisms maps such that if $\mathscr{S}$ and $\mathscr{S}'$ are logical systems, then a map between them is a quadruple $(\Phi, \alpha, \beta, \gamma) : \mathscr{S} \longrightarrow \mathscr{S}'$ such that

$$\text{logic}(\mathscr{S}) \xrightarrow{(\Phi,\alpha,\beta)} \text{logic}(\mathscr{S}')$$

maps the logic component of $\mathscr{S}$ to its counterpart in $\mathscr{S}'$ and

$$\text{pcalc}(\mathscr{S}) \xrightarrow{(\Phi,\alpha,\gamma)} \text{pcalc}(\mathscr{S}')$$

similarly maps the proof calculus of $\mathscr{S}$ to the proof calculus of $\mathscr{S}'$. Here, logic and pcalc are projection operations having the obvious definition—these have been introduced merely for notational convenience. □

4.10 Framework Logics

This chapter has first and foremost shown that we may represent logics and logical systems in the categorical context. The primary strength of these representations is that one may in a rather convenient manner move from one representation to another. The running example having been the transition between equational logic and first order logic. While the construction of the various morphisms in this example has been relatively straight-forward, this may not always be the case. In fact, as one might imagine, they can be considerably harder. This problem is further compounded by the continuous creation of new logics as research into new areas often give rise to the need of new representations and notions of logic. For each new logic created, there exists a need to identify and construct mappings from the new logic to the old familiar ones; certainly not a trivial problem.

These difficulties form to a large part the motivation behind the notion of a *framework logic*. That is, very general style of logic in which many other logics may be represented. This allows the framework logic to act as a central hub when translating between logics, that is, if one with to go from a newly created logic to various other logics it is possible to first translate it into the framework logic and then use one of the—hopefully existing—translation from the framework logic to the target logics. The benefit is obvious, we save ourselves a lot of work by only having to specify a small number of translations. The problem is also obvious, a framework logic would have to be very general indeed before it is practically useful. Ideally, one would like the framework logic to be general enough to represent most logics of interest—known and unknown—while at the same time being manageable in size and complexity. A few candidates have been proposed, see e.g. [35]. One which has received quite some attention is *rewriting logic* which was introduced by Narciso Martí-Oliet and José Meseguer [30] and as we shall see in the following chapter, rewriting logic has been quite successfully put into practical use.

Chapter 5

Applications

The focus of this thesis has been the theoretic notions of logic and how these are readily adaptable for a description based on the framework of category theory. The pursuit for such a description is not motivated solely by intellectual curiosity, it is in fact quite possible to apply these notions in more practical settings. Since, as mentioned, the focus has been on the theoretical foundations only two such settings will be considered here and those will only be discussed in a brief and superficial manner. For more in-depth coverage of these, please see the references given. The two problem settings covered is first, the problem of providing decision support systems in medical situations and second, the problem of formal program specifications—in the form of algebraic specifications.

5.1 Medical Decision Support Systems

In the field of medicine, the concepts presented in this thesis may find themselves in an important role. In particular, research on medical decision support systems has moved in the direction of basing these systems on the notions of formal logic [26, 41]. This approach stems from the observation that *clinical guidelines* which provide physicians with a uniform set of rules for finding a correct diagnosis and cost-efficient treatment [41]. The connection with formal logic—in the manner which we have examined—may easily be found by considering a guideline as a logic having as axioms the guideline rules. Using these rules—or axioms—together with a set of symptoms, a diagnosis may then be found using a deductive process. Given a suitable mechanization of this deductive process it becomes possible to construct a decision support system which help a physician efficiently diagnose a particular patient.

In [11, 26], preliminary work in the construction of formal representations of a few particular guidelines was presented. These representations take the form of two-valued and many-valued logics and this together with the fact that different guidelines may be needed when performing the diagnostic deduction, present a problem for which the notions of Chapter 4 seem perfectly suited. Using the idea of institutions, entailment systems, and so forth, a consistent map between them would allow an integrated reasoning process which take into account all necessary guidelines. The consistency of this map is of course essential—a faulty diagnosis could prove catastrophic—and the guarantees given us by the categorical approach are therefore very tantalizing. This direction of research will be presented in greater depth by Helena Lindgren in [25].

5.2 Algebraic Specifications

In essence the goal of *algebraic specifications* is to be able to reason in a formal manner about the correctness of a program with regard to its expected functionality. By formally specifying program functionality it is possible to more easily experiment with the program design and eventually aid in the construction of the program itself—it may even be possible to automatically construct the program [44]. In practice, this is done with the help of *term rewriting systems* or *specification languages*. In this type of language or system we are, first, able to specify the sorts and operations of the program and, second, the axioms which must be fulfilled by the program [40]. This situation is of course directly analogous to our, by now familiar, notions of signatures and theories.

One of the most popular languages which—among other things—provide facilities for algebraic specification is Maude [6] system. This system is based on the previously mentioned rewriting logic and in its operation it uses the categorical view of logic we have described. As a result, one is able to in the Maude language construct modules which specify maps taking a given logic into rewriting logic. In [30], such a map is given in detail for linear logic [13]. By being allowed to create such maps we are, using a single software system, able to work with many different logics. This is of course very convenient.

To very briefly familiarize ourselves with the functionality of this type of algebraic specification systems, we may consider the theory $T_{EqNatList}$ of Example 34. Fortunately, it is possible to construct this example directly as a rewriting theory since equational logic is a part of rewriting logic [6]. Note that we, using the keyword `protecting`, import the build-in modules `NAT` and `BOOL` which provide natural numbers and booleans, respectively. This import also include the operations acting on the `Nat` and `Bool` sorts. The importing is somewhat analogous to the situation in Example 34. The Maude module for $T_{EqNatList}$ is

```
fmod NATLIST is
    protecting NAT .     protecting BOOL .
    sort Natlist .
    op nil : -> Natlist .
    op head _ : Natlist -> Nat .
    op tail _ : Natlist -> Natlist .
    op cons _ _ : Nat Natlist -> Natlist .
    op empty _ : Natlist -> Bool .
    vars x : Nat .       vars y : Natlist .
    eq empty nil = true .
    eq empty (cons x y) = false .
    eq head (cons x y) = x .
    eq tail (cons x y) = y .
endfm
```

With this module specified, it is now possible to evaluate $\Sigma_{\mathrm{NATLIST}}$-terms in the $T_{EqNatList}$ theory using the `reduce` keyword. A slightly abbreviated session may for example give

```
Maude> reduce in NATLIST : head (tail (cons 3 (cons 2 (cons 1 nil)))) .
result NzNat: 2
Maude> reduce in NATLIST : empty (tail (cons 1 nil)) .
result Bool: true
```

A more in-depth treatment of the construction of specification languages may be found in the excellent book *Specification of Abstract Data Types* by Loeckx, Ehrich, and Wolf [27]. The book builds from the very basics of signatures, algebras and models—as we have done in throughout this text—into a fairly featureful formal specification language called *SL*.

Chapter 6

Discussion and Conclusions

As was mentioned in the introduction, a considerable amount of ground has been covered in this thesis. In addition to this, many of the concepts involved are quite intricate and one may require extensive pondering before fully grasping them—at least this has been my experience. Despite this, I hope that much of it has proven interesting and thought provoking. It is clear that the implications of being able to jump from logic to logic while retaining the semantics are of considerable interest and the possibilities are perhaps far more extensive than one might suspect. In many ways this is still a very fresh field of research so much is still left to discover. Even basic issues such as notation and foundational definitions are still in something of a state of flux. For this reason, a reader comparing this thesis with other texts on the same subject might notice details differing regarding notation and definitions. For the most part, the notation has in this text been made more explicit—especially compared to the important source text of [31], which although supremely elegant may seem somewhat obtuse. This, more explicit, notation was chosen quite carefully and hopefully will help make the subject matter somewhat more accessible for the unfamiliar reader.

The examples given in the thesis have primarily focused on a many-sorted type of equational and first-order logic, respectively. The motivation behind limiting the examples to these two cases is that it allows more in-depth consideration of the mechanics and mathematics behind the notions presented without continually having to bring up unrelated concepts. Unfortunately, this limitation may give the appearance that institutions, entailment systems, and so on are primarily intended to represent these logics. But, this is certainly not the case! One should therefore take care so as to not think of these notions as being limited to the cases shown, they are in fact extremely general and will allow the description of many different—and practically useful—logics.

An interesting aspect of the mappings which have been covered is that it is often possible to describe, in a consistent manner, maps from a “rich” logic to one which is less rich. In Example 39 such a map was given between the institution of many-sorted equational logic to its unsorted cousin. It was there commented that these maps will be lossy, however, one can imagine situations where this is acceptable and allows a more practically useful logic. That is, if the logic of interest is of an extraordinarily complex nature then an acceptable map to a more tractable logic gives the opportunity to draw useful conclusions in an efficient manner. The crucial observation to make is that the maps described *guarantee* the invariance of truth. In other words, we may be certain that the conclusions drawn in the simplified logic are also valid in the more complex one.

Before closing this discussion, it is worth mentioning a final interesting aspect of the notion of institutions, proof calculi, and their cousins. Namely the idea of "borrowing logic". In essence this means that given a logic which lack, for example, a proof calculi it is possible to adapt the proof calculi of a different logic provided a suitable map can be found. Unfortunately, there is no room for further discussion of this fascinating issue. Instead, the interested reader is referred to the fine—and aptly named—article *May I Borrow Your Logic?* by Maura Cerioli and José Meseguer.

6.1 Future Work

During the preparation of this thesis it has become apparent that the content covered is part of an active and vibrant field of studies, one in which there still are many open questions and many important tasks still to perform. An attempt at covering all of them in this section of course would be futile—that could in itself quite likely be a rather important contribution to the field and possibly an interesting topic for a master's thesis. Instead, two widely separate possible future directions are discussed. First, the possibility of generalizing our notion of terms and second, the popularization of the subject. These issues may or may not be suitable topics for further study within the boundaries of future thesis work for interested students.

6.1.1 Generalization Using Monads

An interesting possibility which was brought to my attention by my supervisor Patrik Eklund is the tentative possibility of extending the notions of logic as we have encountered them in Chapter 4 into a far more general concept with the help of monads. I will here briefly expand on this idea. Since they have previously only been superficially mentioned in Chapter 2, it may be of interest to give a quick definition of monads.

Definition 64 (Monads) A monad $\mathbf{F}$ over some category $\mathtt{C}$ is a triple $\mathbf{F} = (F, \eta, \mu)$ where $F : \mathtt{C} \longrightarrow \mathtt{C}$ is a functor and the natural transformations $\eta : \mathrm{id} \longrightarrow F$ and $\mu : F \circ F \longrightarrow F$ are such that $\mu \circ F\mu = \mu \circ \mu F$ and $\mu \circ F\eta = \mu \circ \eta F = \mathrm{id}_F$. □

Before expanding on the use of monads in the generalization hinted at above we must recall from Definition 20 that $T_\Sigma X$ represents the family of Σ-terms using the associated family of variables X. While it was omitted at that time, it is in fact possible to extend T_Σ to a functor $T_\Sigma : \mathtt{FSet} \longrightarrow \mathtt{FSet}$, where $\mathtt{FSet}$ is the category of families of sets, which takes a family of Σ-variables to the family of terms over the signature Σ using the given variables. Furthermore, as shown in [10], T_Σ may be viewed as a monad $\mathbf{T}_\Sigma = (T_\Sigma, \eta^{T_\Sigma}, \mu^{T_\Sigma})$—where η^{T_Σ} and μ^{T_Σ} are suitably defined natural transformations.

Using the term monad it is now possible to view, for example, an entailment system not as the triple $(\mathtt{Sign}, Sen, \vdash)$ of Definition 42, but rather a triple $(\mathbf{T}_\Sigma, Sen, \vdash)$ where the *Sen* functor of this new entailment system structure is a slight modification of the old *Sen* functor. Since variable substitution is an essential part of the *Sen* functors we have viewed so far, it is worth nothing that η^{T_Σ}, by Definition 64, has precisely the appearance of a variable substitution—it is in fact the identity inclusion substitution. However, since *all* monads has a natural transformation with this appearance would it not be possible to generalize the entailment system to work with *any* monad? It does not seem entirely unfeasible since there must, at the very least, be one variable substitution we may use in *Sen*. In other words, is $(\mathbf{F}, Sen, \vdash)$, for any monad $\mathbf{F}$, an entailment system?

Furthermore, monads have been shown to be composable under certain conditions, see e.g. [12]. A question then arise: Using the monad composition $\mathbf{F} \circ \mathbf{G}$, where $\mathbf{F}$ and $\mathbf{G}$ are any composable monads, what would the properties of the entailment system $(\mathbf{F} \circ \mathbf{G}, Sen, \vdash)$ be? How should maps of these entailment systems be defined, that is, which conditions must be upheld?

These ideas represent interesting and currently poorly understood possibilities of generalizing the structures presented in Chapter 4. Since the idea is poorly understood it is at this point hard to tell what the implications of finding the conditions necessary for constructing consistent structures and maps—indeed, if such conditions even exists. Subsequently, very interesting possibilities for foundational research exists in the area.

6.1.2 Implementation of Decision Support System

In Section 5.1, it was explained how there exists a tantalizing possibility of using institutions, entailment systems, and so on, in a formal representation of clinical guidelines. As was mentioned, this construction would allow automated reasoning to be used in a decision support system. As was also mentioned, however, this approach has not been thoroughly examined and a considerable amount of interesting research remains—both in the area of foundational theory and in how to best apply it in a clinical setting. The open questions range from the problem of finding a good formal representation of clinical guidelines—see e.g. [11] for some previous work in this area—to the problem of deciding how to best present a system based on these notions to a working physician.

6.1.3 Popularization

As we have seen, and the discussion on applications have provided hints towards, the ideas described in this thesis have great worth. Both as a field of theoretical study and for more hands-on application. However, today much of the information on the subject is only to be found in research papers and as a result is often rather obtuse and extremely technical for the non-research oriented reader. This text has to a great part concentrated on presenting the notions in a less daunting fashion—whether this aim has been successful or not is of course up to the reader. In either case, it is likely that there is room for a more accessible presentation of the subject. Of course, correctness should not be compromised but there would not be a need to be quite as rigorous as has been the case in this thesis. This more accessible presentation could, perhaps, take the form of a book aimed at undergraduate studies, interested enthusiasts, and members of the industry. One could even—in a fit of splendid naïvité—imagine adapting of some of the topics covered into a form suitable for inclusion in a popular scientific book on theoretical computing science.

6.2 Acknowledgments

This master's thesis would never have seen the light of day had it not been for Patrik Eklund. As my thesis supervisor, his support, enthusiasm and extensive knowledge in what seems like all aspects of theoretical computing science made the work of authoring this text into a joyous one. I am also very grateful to Stephen Hegner whose fine suggestions helped improve this thesis.

Finally, I would like to thank Matilda for her endless patience with my endless writing.

References

[1] Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and concrete categories*. Wiley-Interscience, New York, NY, USA, 1990.

[2] Michael Barr and Charles Wells. *Category theory for computing science*. Prentice-Hall, Inc., 1990.

[3] J. L. Bell. Category Theory and the Foundations of Mathematics. *British Journal for the Philosophy of Science*, 32(4):349–358, December 1981.

[4] Stanley Burris and H.P. Sankappanavar. *A Course in Universal Algebra*. Springer-Verlag, 1981.

[5] Alessandro Cimatti, Fausto Giunchiglia, and Richard W. Weyhrauch. A many-sorted natural deduction. *Computational Intelligence*, 14:134–149, 1998.

[6] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 2001.

[7] Anton Dumitriu. *History of Logic*, volume IV. Abacus Press, 1977.

[8] David S. Dummit and Richard M. Foote. *Abstract Algebra*. Wiley, 4 edition, 2004.

[9] Samuel Eilberg and Saunders Mac Lane. General theory of natural equivalences. *Trans. Amer. Math. Soc.*, 58:231–244, 1945.

[10] P. Eklund, M. A. Galán, M. Ojeda-Aciego, and A. Valverde. Set functors and generalised terms. In *Proc. of IPMU 2000*, volume III of *Information Processing and Management of Uncertainty in Knowledge-based Systems*, pages 1595–1599, 2000.

[11] P. Eklund and H. Lindgren. Towards dementia diagnosis logic. In *Proc. of IPMU 2006*, Information Processing and Management of Uncertainty in Knowledge-based Systems, pages 57–66, 2006.

[12] M.A. Galán. *Categorical Unification*. PhD thesis, Umeå University, 2004.

[13] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[14] Joseph A. Goguen. What is unification? A categorical view of substitution, equation and solution. In Maurice Nivat and Hassan Aït-Kaci, editors, *Resolution of Equations in Algebraic Structures, Volume 1: Algebraic Techniques*, pages 217–261. Academic Press, 1989.

[15] Joseph A. Goguen. A Categorical Manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67, March 1991.

[16] Joseph A. Goguen and Rod M. Burstall. Introducing institutions. In *Proceedings of the Carnegie Mellon Workshop on Logic of Programs*, pages 221–256, London, UK, 1984. Springer-Verlag.

[17] Joseph A. Goguen and Rod M. Burstall. INSTITUTIONS: Abstract Model Theory for Specification and Programming. *J. ACM*, 39(1):95–146, 1992.

[18] Joseph A. Goguen and José Meseguer. Completeness of many-sorted equational logic. *SIGPLAN Not.*, 17(1):9–17, 1982.

[19] Joseph A. Goguen and José Meseguer. Remarks on remarks on many-sorted equational logic. *SIGPLAN Not.*, 22(4):41–48, 1987.

[20] Joseph A. Goguen and Jose Meseguer. Order-sorted algebra i: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.

[21] Ralph P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction*. Addison-Wesley, 1999.

[22] John L. Kelley. *General Topology*. Van Nostrand, 1955.

[23] Claude Kirchner and Hélène Kirchner. *Rewriting Solving Proving*. Preliminary version, January 2006. Available from `http://www.loria.fr/~ckirchne/=rsp/rsp.pdf`.

[24] Elaine Landry and Jean-Pierre Marquis. Categories in Context: Historical, Foundational, and Philosophical. *Philosophia Mathematica*, 13(1):1–43, 2005.

[25] Helena Lindgren. Towards formalizing clinical investigation of dementia using institutions. Unpublished draft.

[26] Helena Lindgren. *Managing knowledge in the development of a decision-support system for the investigation of dementia*. Umeå University, 2005.

[27] Jacques Loeckx, Hans-Dieter Ehrich, and Markus Wolf. *Specification of Abstract Data Types*. Wiley-Teubner, 1996.

[28] Saunders Mac Lane. *Categories for the Working Mathematician (Graduate Texts in Mathematics)*. Springer, September 1998.

[29] Ernest G. Manes. *Algebraic Theories*, volume 26 of *Graduate Texts in Mathematics*. Springer-Verlag, 1976.

[30] Narciso Martí-Oliet and José Meseguer. Rewriting Logic as a Logical and Semantic Framework. *Electronic Notes in Theoretical Computer Science*, 4, 1996.

[31] José Meseguer. General logics. In H.-D. Ebbinghaus, editor, *Logic Colloquium '87*, pages 275–329, Granada, Spain, July 1989. North-Holland.

[32] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings 4th Annual IEEE Symp. on Logic in Computer Science, LICS'89, Pacific Grove, CA, USA, 5–8 June 1989*, pages 14–23. IEEE Computer Society Press, Washington, DC, 1989.

[33] Ben-Ari Mordechai. *Mathematical Logic for Computer Science*. Springer Verlag, 2nd edition, 2001.

[34] F. J. Pelletier. A Brief History of Natural Deduction. *History and Philosophy of Logic*, 20:1–31, 1999.

[35] Frank Pfenning. Logical frameworks. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1063–1147. Elsevier and MIT Press, 2001.

[36] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, 1991.

[37] Dag Prawitz. *Natural Deduction: A proof-theoretical study*. Almqvist & Wiksell, Stockholm, 1965.

[38] Henry G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[39] David E. Rydeheard and Rod M. Burstall. *Computational Category Theory*. Prentice-Hall, Inc., 1988.

[40] Donald Sannella and Andrzej Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, 9(3):229–269, 1997.

[41] Richard N. Shiffman. Representation of clinical practice guidelines in conventional and augmented decision tables. *J. Am. Med. Inform. Assoc.*, 4(5):382–393, 1997.

[42] Lynn Arthur Steen and J. Arthur Seebach Jr. *Counterexamples in Topology*. Courier Dover Publications, September 1995.

[43] Philip Wadler. Comprehending monads. In *Mathematical Structures in Computer Science, Special issue of selected papers from 6'th Conference on Lisp and Functional Programming*, number 2, pages 461–493, 1992.

[44] H. R. Walters. *On Equal Terms - Implementing Algebraic Specifications*. PhD thesis, University of Amsterdam, Amsterdam, 1991.

[45] Lu Zhongwan. *Mathematical Logic for Computer Science*. World Scientific Publishing Co. Pte. Ltd., P O Box 128, Farrer Road, Singapore 9128, 1989.