Donald Sannella and Andrzej Tarlecki

Foundations of Algebraic Specification and Formal Software Development

September 29, 2010

Springer

Contents

Chapter 1
Universal algebra

The most basic assumption of work on algebraic specification is that programs are modelled as algebras. This point of view abstracts from the concrete details of code and algorithms, and regards the input/output behaviour of functions and the representation of data as primary. Representing programs in terms of sets (of data values) and ordinary mathematical functions over these sets greatly simplifies the task of reasoning about program correctness. See Section 0.1 for some illustrative examples and more introductory discussion on this point.

The branch of mathematics that deals with algebras in this general sense (as opposed to the study of specific classes of algebras, such as groups and rings) is called *universal algebra* or sometimes *general algebra*. However, work on universal algebra by mathematicians has concentrated almost exclusively on the special case of single-sorted algebras with first-order total functions. The generalisation to *many-sorted* or *heterogeneous* algebras is required to model programs that manipulate several kinds or *sorts* of data; further generalisations are necessary to handle programs that fail to terminate on some inputs, that generate exceptions during execution, etc. This chapter summarizes the basic concepts and results of many-sorted universal algebra that will be required for the rest of this book. Some extensions useful for modelling more complex programs will be discussed later, in Section 2.7. In this chapter, all proofs are left as exercises for the reader.

1.1 Many-sorted sets

When using an algebra to model a program which manipulates several sorts of data, it is natural to partition the underlying set of values in the algebra so that there is one set of values for each sort of data. It is often convenient to manipulate such a family of sets as a unit, in such a way that operations on this unit respect the "typing" of data values.

The following sequence of definitions and notational conventions allow us to manipulate sorted families of sets (of functions, of relations, ...) in just the same way

as ordinary sets (functions, relations, ...). Ordinary sets (functions, relations, ...) correspond to the degenerate case in which there is just one sort, so these definitions also serve to recall the notation and terminology of set theory to be used throughout this book. Let S be a set; the notation $\langle X_s\rangle_{s\in S}$ is a standard shorthand for the family of objects X_s indexed by $s \in S$, i.e. the function with domain S which maps each $s \in S$ to X_s.

Throughout this section, let S be a set (of sorts).

Definition 1.1.1 (Many-sorted set). An *S-sorted set* is an S-indexed family of sets $X = \langle X_s\rangle_{s\in S}$, which is *empty* if X_s is empty for all $s \in S$. The empty S-sorted set will be written (ambiguously) as $\varnothing$. The S-sorted set X is *finite* if X_s is finite for all $s \in S$ and there is a finite set $\widehat{S} \subseteq S$ such that $X_s = \varnothing$ for all $s \in S \setminus \widehat{S}$.

Let $X = \langle X_s\rangle_{s\in S}$ and $Y = \langle Y_s\rangle_{s\in S}$ be S-sorted sets. Union, intersection, Cartesian product, disjoint union, inclusion (subset) and equality of X and Y are defined componentwise as follows:

$X \cup Y = \langle X_s \cup Y_s\rangle_{s\in S}$
$X \cap Y = \langle X_s \cap Y_s\rangle_{s\in S}$
$X \times Y = \langle X_s \times Y_s\rangle_{s\in S}$
$X \uplus Y = \langle X_s \uplus Y_s\rangle_{s\in S}$ (where $X_s \uplus Y_s = (\{1\} \times X_s) \cup (\{2\} \times Y_s)$)
$X \subseteq Y$ iff (if and only if) $X_s \subseteq Y_s$ for all $s \in S$
$X = Y$ iff $X \subseteq Y$ and $Y \subseteq X$ (equivalently, iff X and Y are equal as functions). □

Exercise 1.1.2. Give a formal explanation of the above statement that "Ordinary sets ... correspond to the degenerate case [of many-sorted sets] in which there is just one sort". How many $\varnothing$-sorted sets are there? □

Notation. It will be very convenient to pretend that $X \subseteq X \uplus Y$ and $Y \subseteq X \uplus Y$. Although this is never actually the case, it allows us to treat disjoint union in the same way as ordinary union, the difference being that when $X \cap Y \neq \varnothing$, $X \uplus Y$ contains two "copies" of the common elements and keeps track of which copy is from X and which from Y. To see that this does not cause problems, observe that there are injective S-sorted functions (see the next definition) $i1\colon X \to X \uplus Y$ and $i2\colon Y \to X \uplus Y$ defined by $i1_s(x) = \langle 1, x\rangle$ for all $s \in S$ and $x \in X_s$ and similarly for $i2$. A pedant would be able to correct what follows by simply inserting the functions $i1$ and/or $i2$ where appropriate in expressions involving $\uplus$. □

Exercise 1.1.3. Extend the above definitions of union, intersection, product and disjoint union to operations on I-indexed families of S-sorted sets, for an arbitrary index set I. For example, the definition for product is $(\prod\langle X_i\rangle_{i\in I})_s = \{\, f\colon I \to \bigcup_{i\in I}(X_i)_s \mid f(i) \in (X_i)_s \text{ for all } i \in I\}$ for each $s \in S$. □

Definition 1.1.4 (Many-sorted function). Let $X = \langle X_s\rangle_{s\in S}$ and $Y = \langle Y_s\rangle_{s\in S}$ be S-sorted sets. An *S-sorted function* $f\colon X \to Y$ is an S-indexed family of functions $f = \langle f_s\colon X_s \to Y_s\rangle_{s\in S}$; X is called the *domain* (or *source*) of f, and Y is called its *codomain* (or *target*). An S-sorted function $f\colon X \to Y$ is an *identity* (an *inclusion*, *surjective*, *injective*, *bijective*, ...) if for every $s \in S$, the function $f_s\colon X_s \to Y_s$ is an identity (an

inclusion, surjective, injective, bijective, ...). The identity S-sorted function on X will be written as $id_X : X \to X$.

If $f : X \to Y$ and $g : Y \to Z$ are S-sorted functions then their *composition* $f;g : X \to Z$ is the S-sorted function defined by $f;g = \langle f_s;g_s \rangle_{s \in S}$. That is, if $s \in S$ and $x \in X_s$ then $(f;g)_s(x) = g_s(f_s(x))$.[1]

Let $f : X \to Y$ be an S-sorted function and $X' \subseteq X$, $Y' \subseteq Y$ be S-sorted sets. The *image of X' under f* is the S-sorted set $f(X') = \langle f_s(X'_s) \rangle_{s \in S} \subseteq Y$, where $f_s(X'_s) = \{ f_s(x) \mid x \in X'_s \} \subseteq Y_s$ for all $s \in S$. The *coimage of Y' under f* is the S-sorted set $f^{-1}(Y') = \langle f_s^{-1}(Y'_s) \rangle_{s \in S} \subseteq X$, where $f_s^{-1}(Y'_s) = \{ x \in X_s \mid f_s(x) \in Y'_s \} \subseteq X_s$ for all $s \in S$. □

Definition 1.1.5 (Many-sorted binary relation). Let $X = \langle X_s \rangle_{s \in S}$ and $Y = \langle Y_s \rangle_{s \in S}$ be S-sorted sets. An *S-sorted binary relation between X and Y*, written $R \subseteq X \times Y$, is an S-indexed family of binary relations $R = \langle R_s \subseteq X_s \times Y_s \rangle_{s \in S}$. For $s \in S$, $x \in X_s$ and $y \in Y_s$, $x \, R_s \, y$ (sometimes written $x \, R \, y$) means $\langle x, y \rangle \in R_s$. □

The generalisation to n-ary relations, for $n \geq 0$, is obvious. As usual, many-sorted functions may be viewed as special many-sorted relations.

Definition 1.1.6 (Kernel of a many-sorted function). Let $f : X \to Y$ be an S-sorted function. The *kernel of f* is the S-sorted binary relation $\ker(f) = \langle \ker(f_s) \rangle_{s \in S} \subseteq X \times X$ where $\ker(f_s) = \{ \langle x, y \rangle \mid x, y \in X_s \text{ and } f_s(x) = f_s(y) \} \subseteq X_s \times X_s$ is the kernel of f_s for all $s \in S$. □

Definition 1.1.7 (Many-sorted equivalence). Let $X = \langle X_s \rangle_{s \in S}$ be an S-sorted set. An S-sorted binary relation $R \subseteq X \times X$ is an *S-sorted equivalence (relation) on X* if it is:

- reflexive: $x R_s x$;
- symmetric: $x R_s y$ implies $y R_s x$; and
- transitive: $x R_s y$ and $y R_s z$ implies $x R_s z$

for all $s \in S$ and $x, y, z \in X_s$. The symbol $\equiv$ is often used for (S-sorted) equivalence relations.

Let $\equiv$ be an S-sorted equivalence on X. If $s \in S$ and $x \in X_s$ then the *equivalence class of x modulo $\equiv$* is the set $[x]_{\equiv_s} = \{ y \in X_s \mid x \equiv_s y \}$. The *quotient of X modulo $\equiv$* is the S-sorted set $X/{\equiv} = \langle X_s/{\equiv_s} \rangle_{s \in S}$ where $X_s/{\equiv_s} = \{ [x]_{\equiv_s} \mid x \in X_s \}$ for all $s \in S$. □

Example 1.1.8. Let $S = \{s_1, s_2\}$, and let X and Y be two S-sorted sets defined as follows:

$X = \langle X_s \rangle_{s \in S}$ where $X_{s_1} = \{\square, \triangle\}$ and $X_{s_2} = \{\clubsuit, \heartsuit, \spadesuit\}$,
$Y = \langle Y_s \rangle_{s \in S}$ where $Y_{s_1} = \{1, 2, 3\}$ and $Y_{s_2} = \{1, 2, 3\}$.

Let $f : X \to Y$ be the S-sorted function such that

[1] This "diagrammatic" order of composition and the semicolon notation will be used consistently throughout this book.

$f_{s_1} = \{\square \mapsto 1, \triangle \mapsto 3\}$
$f_{s_2} = \{\clubsuit \mapsto 1, \heartsuit \mapsto 2, \spadesuit \mapsto 2\}$.

(i.e., $f_{s_1}(\square) = 1$ and $f_{s_1}(\triangle) = 3$; analogously for f_{s_2}). Then the kernel of f is the S-sorted equivalence relation $\ker(f) = \langle \ker(f_s) \rangle_{s \in S}$ where

$\ker(f_{s_1}) = \{\langle \square, \square \rangle, \langle \triangle, \triangle \rangle\}$
$\ker(f_{s_2}) = \{\langle \clubsuit, \clubsuit \rangle, \langle \heartsuit, \heartsuit \rangle, \langle \heartsuit, \spadesuit \rangle, \langle \spadesuit, \heartsuit \rangle, \langle \spadesuit, \spadesuit \rangle\}$.

The quotient of X modulo $\ker(f)$ is the S-sorted set $X/\ker(f) = \langle X_s/\ker(f_s) \rangle_{s \in S}$ where

$X_{s_1}/\ker(f_{s_1}) = \{\{\square\}, \{\triangle\}\}$
$X_{s_2}/\ker(f_{s_2}) = \{\{\clubsuit\}, \{\heartsuit, \spadesuit\}\}$. □

Exercise 1.1.9. Show that if $f : X \to Y$ is an S-sorted function, then $\ker(f)$ is an S-sorted equivalence on X. □

Exercise 1.1.10. Show that if $\equiv$ is an S-sorted equivalence on X then for all $s \in S$ and $x, y \in X_s$, $[x]_{\equiv_s} = [y]_{\equiv_s}$ iff $x \equiv_s y$. □

Notation. Subscripts selecting components of S-sorted sets (functions, relations, ...) are often omitted when there is no danger of confusion. Then Exercise 1.1.10 would read: "... for all $s \in S$ and $x, y \in X_s$, $[x]_{\equiv} = [y]_{\equiv}$ iff $x \equiv y$." □

1.2 Signatures and algebras

The functions and data types defined by a program have names. These names are used to compute with and reason about the program, and to build larger programs which rely on the functionality the program provides. The connection between a program and an algebra used to model it is provided by these names, which are attached to the corresponding components of the algebra. The set of names associated with an algebra is called its signature. The signature of an algebra defines the *syntax* of the algebra by characterising the ways in which its components may legally be combined; the algebra itself supplies the *semantics* by assigning interpretations to the names in the signature.

Definition 1.2.1 (Many-sorted signature). A *(many-sorted) signature* is a pair $\Sigma = \langle S, \Omega \rangle$, where:

- S is a set (of sort names); and
- Ω is an $S^* \times S$-sorted set (of operation names)

where S^* is the set of finite (including empty) sequences of elements of S. We will sometimes write $sorts(\Sigma)$ for S and $ops(\Sigma)$ for Ω. Σ is a *subsignature* of a signature $\Sigma' = \langle S', \Omega' \rangle$ if $S \subseteq S'$ and $\Omega_{w,s} \subseteq \Omega'_{w,s}$ for all $w \in S^*, s \in S$. □

Many-sorted signatures will be referred to as *algebraic* signatures when it is necessary to distinguish them from other kinds of signatures to be introduced later.

Notation. Saying that $f: s_1 \times \cdots \times s_n \to s$ is in $\Sigma = \langle S, \Omega \rangle$ means that $s_1 \ldots s_n \in S^*$, $s \in S$ and $f \in \Omega_{s_1 \ldots s_n, s}$. Then f is said to have *arity* $s_1 \ldots s_n$ and *result sort* s. The abbreviation $f: s$ will be used for $f: \varepsilon \to s$ (ε is the empty sequence). □

This definition of signature does not accommodate programs containing higher-order functions, or functions returning multiple results. A possible extension to handle higher-order functions is briefly discussed in Section 2.7.6. As for functions with multiple results, a function $f: s_1 \times \cdots \times s_n \to t_1 \times \cdots \times t_m$ may be viewed as a family of m functions

$$f_1: s_1 \times \cdots \times s_n \to t_1 \qquad \ldots \qquad f_m: s_1 \times \cdots \times s_n \to t_m.$$

Generalising the definition of signature to handle such functions in a more direct way is easy but makes subsequent developments somewhat messier in a non-interesting way.

The definition above *does* permit overloaded operation names, since it is possible to have both $f: s_1 \times \cdots \times s_n \to s$ and $f: t_1 \times \cdots \times t_m \to t$ in a signature Σ, where $s_1 \ldots s_n s \neq t_1 \ldots t_m t$. A more restrictive definition of signature, adequate for most purposes, would have a set Ω of operation names (and a set S of sort names) with functions $\mathit{arity}: \Omega \to S^*$ and $\mathit{sort}: \Omega \to S$. These two definitions are equivalent if each operation name in Ω is taken to be tagged with its arity and result sort.

In the rest of this section, let $\Sigma = \langle S, \Omega \rangle$ be a signature.

Definition 1.2.2 (Many-sorted algebra). A Σ-*algebra* A consists of:

- an S-sorted set $|A|$ of *carrier sets* (or *carriers*); and
- for each $f: s_1 \times \cdots \times s_n \to s$ in Σ, a function (or *operation*) $(f: s_1 \times \cdots \times s_n \to s)_A: |A|_{s_1} \times \cdots \times |A|_{s_n} \to |A|_s$. □

If A is a Σ-algebra and s is a sort name in Σ then $|A|_s$, the carrier set of sort s in A, is the universe of data values of sort s; accordingly, we often refer to the elements of carrier sets as *values*. If $f: s_1 \times \cdots \times s_n \to s$ is in Σ then the operation $(f: s_1 \times \cdots \times s_n \to s)_A$ is a function on the corresponding carrier sets of A. If $n = 0$ (i.e. $f: s$), then $|A|_{s_1} \times \cdots \times |A|_{s_n}$ is a singleton set containing the empty tuple $\langle\rangle$, and then $(f: s)_A$ may be viewed as a constant denoting the value $(f: s)_A(\langle\rangle) \in |A|_s$. Notice that $(f: s_1 \times \cdots \times s_n \to s)_A$ is a *total* function[2] so algebras as defined here are only appropriate for modelling programs containing total functions. See Sections 2.7.3–2.7.5 for several ways of extending the definitions to cope with partial functions. Note also that there is no restriction on the cardinality of $|A|_s$; in particular, $|A|_s$ may be empty and need not be countable.

Notation. Let A be a Σ-algebra and let $f: s_1 \times \cdots \times s_n \to s$ be in Σ. We always write f_A in place of $(f: s_1 \times \cdots \times s_n \to s)_A$ when there is no danger of confusion. When $n = 0$ (i.e. $f: s$), we write $(f: s)_A$ or f_A in place of $(f: s)_A(\langle\rangle)$. □

[2] All functions in this book are total except where they are explicitly designated as partial.

Exercise 1.2.3. If $\Omega_{\varepsilon,s} \neq \varnothing$ for some $s \in S$, then there are no $\langle S, \Omega \rangle$-algebras having an empty carrier of sort s. Characterise signatures for which all algebras have non-empty carriers of all sorts. □

Example 1.2.4. Let $S1 = \{\mathit{shape}, \mathit{suit}\}$ and let $\Omega 1_{\varepsilon,\mathit{shape}} = \{\mathit{box}\}$, $\Omega 1_{\varepsilon,\mathit{suit}} = \{\mathit{hearts}\}$, $\Omega 1_{\mathit{shape},\mathit{shape}} = \{\mathit{boxify}\}$, $\Omega 1_{\mathit{shape}\,\mathit{suit},\mathit{suit}} = \{f\}$, and $\Omega 1_{w,s} = \varnothing$ for all other $w \in S1^*, s \in S1$. Then $\Sigma 1 = \langle S1, \Omega 1 \rangle$ is a signature with sort names *shape* and *suit* and operation names $\mathit{box}\colon \mathit{shape}$, $\mathit{hearts}\colon \mathit{suit}$, $\mathit{boxify}\colon \mathit{shape} \to \mathit{shape}$ and $f\colon \mathit{shape} \times \mathit{suit} \to \mathit{suit}$. We can present $\Sigma 1$ in tabular form as follows (this notation will be used later with the obvious meaning):

$$\begin{array}{lll} \Sigma 1 = & \textbf{sorts} & \mathit{shape}, \mathit{suit} \\ & \textbf{ops} & \mathit{box}\colon \mathit{shape} \\ & & \mathit{hearts}\colon \mathit{suit} \\ & & \mathit{boxify}\colon \mathit{shape} \to \mathit{shape} \\ & & f\colon \mathit{shape} \times \mathit{suit} \to \mathit{suit} \end{array}$$

We define a $\Sigma 1$-algebra $A1$ as follows:

$$\begin{array}{l} |A1|_{\mathit{shape}} = \{\square, \triangle\}, \\ |A1|_{\mathit{suit}} = \{\clubsuit, \heartsuit, \spadesuit\}, \\ \mathit{box}_{A1} = \square \in |A1|_{\mathit{shape}}, \\ \mathit{hearts}_{A1} = \heartsuit \in |A1|_{\mathit{suit}}, \\ \mathit{boxify}_{A1}\colon |A1|_{\mathit{shape}} \to |A1|_{\mathit{shape}} = \{\square \mapsto \square, \triangle \mapsto \square\}, \end{array}$$

and $f_{A1}\colon |A1|_{\mathit{shape}} \times |A1|_{\mathit{suit}} \to |A1|_{\mathit{suit}}$ is defined by the following table:

f_{A1}	$\clubsuit$	$\heartsuit$	$\spadesuit$
$\square$	$\clubsuit$	$\spadesuit$	$\heartsuit$
$\triangle$	$\heartsuit$	$\spadesuit$	$\spadesuit$

(NOTE: Reference will be made to $\Sigma 1$ and $A1$ in examples throughout the rest of this chapter.) □

Definition 1.2.5 (Subalgebra). Let A and B be Σ-algebras. B is a *subalgebra* of A if:

- $|B| \subseteq |A|$; and
- for $f\colon s_1 \times \cdots \times s_n \to s$ in Σ and $b_1 \in |B|_{s_1}, \ldots, b_n \in |B|_{s_n}$, $f_B(b_1, \ldots, b_n) = f_A(b_1, \ldots, b_n)$.

B is a *proper* subalgebra of A if it is a subalgebra of A and $|B| \neq |A|$. A subalgebra of A is determined by an S-sorted subset $|B|$ of $|A|$ which is closed under the operations of Σ, i.e. such that for each $f\colon s_1 \times \cdots \times s_n \to s$ in Σ and $b_1 \in |B|_{s_1}, \ldots, b_n \in |B|_{s_n}$, $f_A(b_1, \ldots, b_n) \in |B|_s$. □

If B is a (proper) subalgebra of A then B is "smaller" than A in the sense that it contains fewer *data values* than A. Both A and B are Σ-algebras though, so A and B contain interpretations for exactly the same sort and operation names.

Exercise 1.2.6. Let A be a Σ-algebra. Show that the intersection of any family of (carriers of) subalgebras of A is a (carrier of a) subalgebra of A. Use this to show that for any $X \subseteq |A|$, there is a least subalgebra of A that contains X. This is called the *subalgebra of A generated by X*. Give an explicit construction of this algebra. (HINT: Consider the family of S-sorted sets $X_i \subseteq |A|$, $i \geq 0$, where $X_0 = X$ and X_{i+1} is obtained from X_i by adding the results of applying the operations of A to arguments in X_i.) □

Definition 1.2.7 (Reachable algebra). Let A be a Σ-algebra. A is *reachable* if A has no proper subalgebra (equivalently, if A is generated by $\varnothing$). □

By Exercise 1.2.6, every algebra has a unique reachable subalgebra.

Example 1.2.8. Let $\Sigma 1 = \langle S1, \Omega 1 \rangle$ and $A1$ be as defined in Example 1.2.4. Define a $\Sigma 1$-algebra $B1$ by

$$
\begin{array}{l}
|B1|_{shape} = \{\Box\}, \\
|B1|_{suit} = \{\heartsuit, \spadesuit\}, \\
box_{B1} = \Box \in |B1|_{shape}, \\
hearts_{B1} = \heartsuit \in |B1|_{suit}, \\
boxify_{B1} : |B1|_{shape} \to |B1|_{shape} = \{\Box \mapsto \Box\}, \\
f_{B1} : |B1|_{shape} \times |B1|_{suit} \to |B1|_{suit} = \{\langle \Box, \heartsuit \rangle \mapsto \spadesuit, \langle \Box, \spadesuit \rangle \mapsto \heartsuit\}.
\end{array}
$$

$B1$ is the subalgebra of $A1$ generated by $\varnothing$. That is, $B1$ is the reachable subalgebra of $A1$. □

Definition 1.2.9 (Product algebra). Let A and B be Σ-algebras. The *product algebra* $A \times B$ is the Σ-algebra defined as follows:

- $|A \times B| = |A| \times |B|$; and
- for each $f : s_1 \times \cdots \times s_n \to s$ in Σ and $\langle a_1, b_1 \rangle \in |A \times B|_{s_1}, \ldots, \langle a_n, b_n \rangle \in |A \times B|_{s_n}$, $f_{A \times B}(\langle a_1, b_1 \rangle, \ldots, \langle a_n, b_n \rangle) = \langle f_A(a_1, \ldots, a_n), f_B(b_1, \ldots, b_n) \rangle \in |A \times B|_s$.

This generalises to the product $\prod \langle A_i \rangle_{i \in I}$ of a family of Σ-algebras, indexed by an arbitrary set I (possibly empty), as follows:

- $|\prod \langle A_i \rangle_{i \in I}| = \prod \langle |A_i| \rangle_{i \in I}$; and
- for each $f : s_1 \times \cdots \times s_n \to s$ in Σ and $f_1 \in |\prod \langle A_i \rangle_{i \in I}|_{s_1}, \ldots, f_n \in |\prod \langle A_i \rangle_{i \in I}|_{s_n}$, $f_{\prod \langle A_i \rangle_{i \in I}}(f_1, \ldots, f_n)(i) = f_{A_i}(f_1(i), \ldots, f_n(i))$ for all $i \in I$. □

Exercise 1.2.10. Definition 1.2.9 shows how two Σ-algebras can be combined to form a new Σ-algebra by taking the Cartesian product of their carriers. According to Exercise 1.2.6, the same thing can be done (with subalgebras of a fixed algebra) using intersection. Try to formulate definitions of *union* and *disjoint union* of algebras, where $|A \cup B| = |A| \cup |B|$ and $|A \uplus B| = |A| \uplus |B|$ respectively. What happens? □

1.3 Homomorphisms and congruences

A homomorphism between algebras is the analogue of a function between sets, and a congruence relation on an algebra is the analogue of an equivalence relation on a set. An algebra has more structure than a set, so homomorphisms and congruences are required to respect the additional structure (i.e. the behaviour of the operations). Homomorphisms and congruences are important basic tools for relating algebras and constructing new algebras from old ones.

Throughout this section, let $\Sigma = \langle S, \Omega \rangle$ be a signature.

Definition 1.3.1 (Homomorphism). Let A and B be Σ-algebras. A *Σ-homomorphism* $h\colon A \to B$ is an S-sorted function $h\colon |A| \to |B|$ which respects the operations of Σ, i.e. such that for all $f\colon s_1 \times \cdots \times s_n \to s$ in Σ and $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$, $h_s(f_A(a_1, \ldots, a_n)) = f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$. A Σ-homomorphism $h\colon A \to B$ is an *identity* (an *inclusion*, *surjective*, ...) if it is an identity (an inclusion, surjective, ...) when viewed as an S-sorted function. □

Notation. If $h\colon A \to B$ is a Σ-homomorphism, then $|h|\colon |A| \to |B|$ denotes h viewed as an S-sorted function. The only difference between h and $|h|$ is that in the case of $|h|$ we have "forgotten" that the additional condition required of a homomorphism is satisfied. □

Informally, the homomorphism condition says that the behaviour of the operations in A is reflected in that of the operations in B. This condition can be expressed in the form of a diagram as follows:

$$\begin{array}{ccc} |A|_{s_1} \times \cdots \times |A|_{s_n} & \xrightarrow{h_{s_1} \times \cdots \times h_{s_n}} & |B|_{s_1} \times \cdots \times |B|_{s_n} \\ \Big\downarrow{\scriptstyle f_A} & & \Big\downarrow{\scriptstyle f_B} \\ |A|_s & \xrightarrow[h_s]{} & |B|_s \end{array}$$

where $(h_{s_1} \times \cdots \times h_{s_n})(a_{s_1}, \ldots, a_{s_n}) = (h_{s_1}(a_{s_1}), \ldots, h_{s_n}(a_{s_n}))$ for all $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$. The homomorphism condition amounts to the requirement that this diagram *commutes*, i.e. that composing the functions on the top and right-hand arrows gives the same result as composing the functions on the left-hand and bottom arrows. Such commuting diagrams will be used heavily in later chapters, particularly in Chapter 3.

Example 1.3.2. Let $\Sigma 1 = \langle S1, \Omega 1 \rangle$ and $A1$ be as defined in Example 1.2.4. Define a $\Sigma 1$-algebra $C1$ by

$|C1|_{shape} = |C1|_{suit} = \{1,2,3\}$,
$box_{C1} = 1 \in |C1|_{shape}$,
$hearts_{C1} = 2 \in |C1|_{suit}$,
$boxify_{C1}: |C1|_{shape} \to |C1|_{shape} = \{1 \mapsto 1, 2 \mapsto 3, 3 \mapsto 1\}$,

and $f_{C1}: |C1|_{shape} \times |C1|_{suit} \to |C1|_{suit}$ is defined by the following table:

f_{C1}	1	2	3
1	1	2	3
2	2	1	2
3	2	2	1

Let $h1: |A1| \to |C1|$ be the $S1$-sorted function such that

$h1_{shape} = \{\square \mapsto 1, \triangle \mapsto 3\}$,
$h1_{suit} = \{\clubsuit \mapsto 1, \heartsuit \mapsto 2, \spadesuit \mapsto 2\}$.

It is easy to verify that $h1: A1 \to C1$ is a $\Sigma 1$-homomorphism by checking the following:

$$\begin{aligned}
h1_{shape}(box_{A1}) &= box_{C1} \\
h1_{suit}(hearts_{A1}) &= hearts_{C1} \\
h1_{shape}(boxify_{A1}(\square)) &= boxify_{C1}(h1_{shape}(\square)) \\
h1_{shape}(boxify_{A1}(\triangle)) &= boxify_{C1}(h1_{shape}(\triangle)) \\
h1_{suit}(f_{A1}(\square, \clubsuit)) &= f_{C1}(h1_{shape}(\square), h1_{suit}(\clubsuit)) \\
h1_{suit}(f_{A1}(\square, \heartsuit)) &= f_{C1}(h1_{shape}(\square), h1_{suit}(\heartsuit)) \\
h1_{suit}(f_{A1}(\square, \spadesuit)) &= f_{C1}(h1_{shape}(\square), h1_{suit}(\spadesuit)) \\
h1_{suit}(f_{A1}(\triangle, \clubsuit)) &= f_{C1}(h1_{shape}(\triangle), h1_{suit}(\clubsuit)) \\
h1_{suit}(f_{A1}(\triangle, \heartsuit)) &= f_{C1}(h1_{shape}(\triangle), h1_{suit}(\heartsuit)) \\
h1_{suit}(f_{A1}(\triangle, \spadesuit)) &= f_{C1}(h1_{shape}(\triangle), h1_{suit}(\spadesuit)).
\end{aligned}$$

□

Exercise 1.3.3. Let A be a Σ-algebra. Show that $id_{|A|}: A \to A$ (the identity S-sorted function) is a Σ-homomorphism. Let $h: A \to B$ and $h': B \to C$ be Σ-homomorphisms. Show that $|h|;|h'|: |A| \to |C|$ is a Σ-homomorphism $h;h': A \to C$. □

Exercise 1.3.4. Let $h: A \to B$ be a Σ-homomorphism, and let A' be a subalgebra of A. Let the *image of A' under h* be the Σ-algebra $h(A')$ defined as follows:

- $|h(A')| = |h|(|A'|)$; and
- for each $f: s_1 \times \cdots \times s_n \to s$ in Σ and $a_1 \in |A'|_{s_1}, \ldots, a_n \in |A'|_{s_n}$, $f_{h(A')}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) = h_s(f_{A'}(a_1, \ldots, a_n))$.

Show that $h(A')$ is a well-defined Σ-algebra (in particular, that the function $f_{h(A')}: |h(A')|_{s_1} \times \cdots \times |h(A')|_{s_n} \to |h(A')|_s$ is well-defined for each $f: s_1 \times \cdots \times s_n \to s$ in Σ) and that it is a subalgebra of B. Formulate a definition of the *coimage* of a subalgebra B' of B under h, and show that it is a subalgebra of A. □

Exercise 1.3.5. Let $h: A \to B$ be a Σ-homomorphism, and suppose $X \subseteq |A|$. Show that the subalgebra of B generated by $|h|(X) \subseteq |B|$ is the image of the subalgebra of A generated by X. Show that it follows that if $h: A \to B$ is surjective and A is reachable then B is reachable. □

Exercise 1.3.6. Let B be a reachable Σ-algebra. Show that for any Σ-algebra A, there is at most one Σ-homomorphism $h\colon B \to A$, and that any Σ-homomorphism $h\colon A \to B$ is surjective. □

Definition 1.3.7 (Isomorphism). Let A and B be Σ-algebras. A Σ-homomorphism $h\colon A \to B$ is a *Σ-isomorphism* if it has an inverse, i.e. there is a Σ-homomorphism $h^{-1}\colon B \to A$ such that $h;h^{-1} = id_{|A|}$ and $h^{-1};h = id_{|B|}$. (**Exercise:** Show that if h^{-1} exists then it is unique.) Then A and B are called *isomorphic* and we write $h\colon A \cong B$ or just $A \cong B$. □

Exercise 1.3.8. Let $h\colon A \cong B$ and $h'\colon B \cong C$ be Σ-isomorphisms. Show that their composition is a Σ-isomorphism $h;h'\colon A \cong C$. Show that $\cong$ (as a binary relation on Σ-algebras) is reflexive and symmetric, and is therefore an equivalence relation. □

Two isomorphic algebras are typically regarded as indistinguishable for all practical purposes. It is easy to see why: the only way in which they can differ is in the particular choice of data values in the carriers. The size of the carriers and the way that the operations behave on the values in the carriers is exactly the same. For this reason we are often satisfied with a definition of an algebra "up to isomorphism", i.e. a description of an isomorphism class of algebras in a context where one would expect a definition of a single algebra. An example of this is in Fact 1.4.10 below. The notion of isomorphism can be generalised to other kinds of structures, where it embodies exactly the same concept of indistinguishability. See Chapter 3 for this generalisation and for many more examples of definitions of objects "up to isomorphism".

Example 1.3.9. Let $\Sigma 1 = \langle S1, \Omega 1\rangle$ and $A1$ be as defined in Example 1.2.4. Define a $\Sigma 1$-algebra $D1$ by

$|D1|_{shape} = \{\square, \triangle\}$,
$|D1|_{suit} = \{1,2,3\}$,
$box_{D1} = \triangle \in |D1|_{shape}$,
$hearts_{D1} = 2 \in |D1|_{suit}$,
$boxify_{D1}\colon |D1|_{shape} \to |D1|_{shape} = \{\square \mapsto \triangle, \triangle \mapsto \triangle\}$,

and $f_{D1}\colon |D1|_{shape} \times |D1|_{suit} \to |D1|_{suit}$ is defined by the following table:

f_{D1}	1	2	3
□	2	3	3
△	1	3	2

Let $i1\colon |A1| \to |D1|$ be the $S1$-sorted function such that

$i1_{shape} = \{\square \mapsto \triangle, \triangle \mapsto \square\}$
$i1_{suit} = \{\clubsuit \mapsto 1, \heartsuit \mapsto 2, \spadesuit \mapsto 3\}$.

This defines a $\Sigma 1$-homomorphism $i1\colon A1 \to D1$ which is a $\Sigma 1$-isomorphism, so $A1 \cong D1$. □

Exercise 1.3.10. Show that a homomorphism is an isomorphism iff it is bijective. □

Exercise 1.3.11. Show that there is an injective homomorphism $h: A \to B$ iff A is isomorphic to a subalgebra of B. □

Example 1.3.12. Let $\Sigma = \langle S, \Omega \rangle$ be the signature

sorts s
ops $a: s$
$f: s \to s$

and define Σ-algebras A and B by

$|A|_s = Nat$ (the natural numbers),
$a_A = 0 \in |A|_s$,
$f_A: |A|_s \to |A|_s = \{n \mapsto n+1 \mid n \in Nat\}$,

$|B|_s = \{n \in Nat \mid$ the Turing machine with Gödel number n halts on all inputs$\}$,
$a_B =$ the smallest $n \in |B|_s$,
$f_B: |B|_s \to |B|_s = \{n \in |B|_s \mapsto$ the smallest $m \in |B|_s$ such that $m > n\}$.

Let $i: |A| \to |B|$ be the S-sorted function such that

$i_s(n) =$ the $(n+1)^{\text{st}}$ smallest element of $|B|_s$

for all $n \in |A|_s$. The function i_s is well-defined since $|B|_s$ is infinite. This defines a Σ-homomorphism $i: A \to B$ which is an isomorphism.

Although $A \cong B$, the Σ-algebras A and B are not "the same" from the point of view of computability: everything in A is computable, in contrast to B ($|B|_s$ is not recursively enumerable and f_B is not computable). Isomorphisms capture *structural* similarity, ignoring what the values in the carriers are and what the operations actually compute. This example shows that, for some purposes, properties stronger than structural similarity are important. □

Definition 1.3.13 (Congruence). Let A be a Σ-algebra. A *Σ-congruence on A* is an (S-sorted) equivalence $\equiv$ on $|A|$ which respects the operations of Σ: for all $f: s_1 \times \cdots \times s_n \to s$ in Σ and $a_1, a_1' \in |A|_{s_1}, \ldots, a_n, a_n' \in |A|_{s_n}$, if $a_1 \equiv_{s_1} a_1'$ and … and $a_n \equiv_{s_n} a_n'$ then $f_A(a_1, \ldots, a_n) \equiv_s f_A(a_1', \ldots, a_n')$. □

Exercise 1.3.14. Show that the intersection of any family of Σ-congruences on A is a Σ-congruence on A. Use this to show that for any S-sorted binary relation R on $|A|$ there is a least (with respect to $\subseteq$) Σ-congruence on A which includes R.

Show that the kernel of any Σ-homomorphism $h: A \to B$ is a Σ-congruence on A.

Show that a surjective Σ-homomorphism is an isomorphism iff its kernel is the identity. □

Definition 1.3.15 (Quotient algebra). Let A be a Σ-algebra, and let $\equiv$ be a Σ-congruence on A. The *quotient algebra of A modulo* $\equiv$ is the Σ-algebra $A/\equiv$ defined by:

- $|A/{\equiv}| = |A|/{\equiv}$; and
- for each $f : s_1 \times \cdots \times s_n \to s$ and $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$, $f_{A/\equiv}([a_1]_{\equiv_{s_1}}, \ldots, [a_n]_{\equiv_{s_n}}) = [f_A(a_1, \ldots, a_n)]_{\equiv_s}$. □

Exercise 1.3.16. Show that $A/{\equiv}$ in Definition 1.3.15 is a well-defined Σ-algebra. □

Example 1.3.17. Let $\Sigma 1 = \langle S1, \Omega 1\rangle$ and $A1$ be as defined in Example 1.2.4, and let $\equiv = \langle \equiv_s \rangle_{s \in S1}$ be the $S1$-sorted binary relation on $|A1|$ defined by

$$\equiv_{shape} = \{\langle \square, \square\rangle, \langle \triangle, \triangle\rangle\}$$
$$\equiv_{suit} = \{\langle \clubsuit, \clubsuit\rangle, \langle \heartsuit, \heartsuit\rangle, \langle \heartsuit, \spadesuit\rangle, \langle \spadesuit, \heartsuit\rangle, \langle \spadesuit, \spadesuit\rangle\}.$$

This defines a congruence on $A1$. $A1/{\equiv}$ is the $\Sigma 1$-algebra defined by

$$|A1/{\equiv}|_{shape} = \{\{\square\}, \{\triangle\}\},$$
$$|A1/{\equiv}|_{suit} = \{\{\clubsuit\}, \{\heartsuit, \spadesuit\}\},$$
$$box_{A1/\equiv} = \{\square\} \in |A1/{\equiv}|_{shape},$$
$$hearts_{A1/\equiv} = \{\heartsuit, \spadesuit\} \in |A1/{\equiv}|_{suit},$$
$$boxify_{A1/\equiv} : |A1/{\equiv}|_{shape} \to |A1/{\equiv}|_{shape} = \{\{\square\} \mapsto \{\square\}, \{\triangle\} \mapsto \{\square\}\},$$

and $f_{A1/\equiv} : |A1/{\equiv}|_{shape} \times |A1/{\equiv}|_{suit} \to |A1/{\equiv}|_{suit}$ is defined by the following table:

$f_{A1/\equiv}$	$\{\clubsuit\}$	$\{\heartsuit, \spadesuit\}$
$\{\square\}$	$\{\clubsuit\}$	$\{\heartsuit, \spadesuit\}$
$\{\triangle\}$	$\{\heartsuit, \spadesuit\}$	$\{\heartsuit, \spadesuit\}$

□

Exercise 1.3.18. Let $\equiv$ be a Σ-congruence on A, and let $h_s(a) = [a]_{\equiv_s}$ for $s \in S$, $a \in |A|_s$. Show that $\langle h_s : |A|_s \to (|A|/{\equiv})_s \rangle_{s \in S}$ is a Σ-homomorphism $h : A \to A/{\equiv}$ with $\ker(h) = \equiv$. □

Exercise 1.3.19. Let $h : A \to B$ be a Σ-homomorphism. Show that $A/\ker(h)$ is isomorphic to $h(A)$. (HINT: The isomorphism is given by $[a]_{\ker(h_s)} \mapsto h_s(a)$ for $s \in S$, $a \in |A|_s$.) □

Exercise 1.3.20. Let $\equiv$ be a Σ-congruence on A. Show that for any Σ-homomorphism $h : A \to B$ such that $\equiv \subseteq \ker(h)$, there exists a unique Σ-homomorphism $g : A/{\equiv} \to B$ such that $h_s(a) = g_s([a]_{\equiv_s})$ for all $s \in S$, $a \in |A|_s$. □

Exercise 1.3.21. Show that there is a surjective homomorphism $h : A \to B$ iff there is a congruence $\equiv$ on A such that B is isomorphic to $A/{\equiv}$. □

Exercise 1.3.22. Let A be a Σ-algebra, let $\equiv$ be a congruence on A and let B be a subalgebra of $A/{\equiv}$. Show that there is a subalgebra C of A and congruence $\equiv'$ on C such that $B = C/{\equiv'}$. □

Exercise 1.3.23. Let $h : A \to B$ be a Σ-homomorphism. Show that there is a unique Σ-congruence $\equiv$ on A and a unique injective Σ-homomorphism $g : A/{\equiv} \to B$ such that $h_s(a) = g_s([a]_{\equiv_s})$ for all $s \in S$, $a \in |A|_s$. □

1.4 Term algebras

For any signature Σ there is a special Σ-algebra whose values are just well-formed terms (i.e. expressions) built from the operation names in Σ. A Σ-algebra of terms with variables is similarly determined by a signature $\Sigma = \langle S, \Omega \rangle$ and an S-sorted set of variables. These algebras are rather boring insofar as modelling programs is concerned — the term algebra models a program which does no real computation. But the homomorphisms from these algebras to *other* algebras turn out to be very useful technical tools, as shown by the definitions below.

Throughout this section, let $\Sigma = \langle S, \Omega \rangle$ be a signature and let X be an S-sorted set (of variables), where $x \in X_s$ for $s \in S$ means that the variable x is of sort s (written $x{:}s$). Note that "overloading" of variable names is permitted here, since there is no requirement that X_s and $X_{s'}$ be disjoint for $s \neq s' \in S$.

Definition 1.4.1 (Term algebra). The *Σ-algebra $T_\Sigma(X)$ of terms with variables X* is the Σ-algebra defined as follows:

- $|T_\Sigma(X)|$ is the least (with respect to $\subseteq$) S-sorted set of words (sequences) over the alphabet
$$S \cup \bigcup_{\substack{w \in S^* \\ s \in S}} \Omega_{w,s} \cup \bigcup_{s \in S} X_s \cup \{:, (, ,,)\}$$
such that:
  - the word "$x{:}s$" $\in |T_\Sigma(X)|_s$ for all $s \in S$ and $x \in X_s$; and
  - for all $f{:}s_1 \times \cdots \times s_n \to s$ in Σ and all words $t_1 \in |T_\Sigma(X)|_{s_1}, \ldots, t_n \in |T_\Sigma(X)|_{s_n}$, the word "$f(t_1,\ldots,t_n){:}s$" $\in |T_\Sigma(X)|_s$.
- for all $f{:}s_1 \times \cdots \times s_n \to s$ in Σ and all words $t_1 \in |T_\Sigma(X)|_{s_1}, \ldots, t_n \in |T_\Sigma(X)|_{s_n}$, $f_{T_\Sigma(X)}(t_1,\ldots,t_n) =$ (the word) "$f(t_1,\ldots,t_n){:}s$" $\in |T_\Sigma(X)|_s$.

(Quotation marks are used here solely to emphasize that terms are words, and are not part of the words they delimit.) If $s \in S$ and $t \in |T_\Sigma(X)|_s$ then t is a *Σ-term of sort s with variables X*; the *free variables of t* is the set $FV(t) \subseteq X$ of variables that actually occur in t: for $s \in S$ and $x \in X_s$, $x \in FV(t)_s$ if t contains the subword "$x{:}s$".

The *Σ-algebra of ground terms* is the Σ-algebra $T_\Sigma = T_\Sigma(\varnothing)$ of terms without variables. If $s \in S$ and $t \in |T_\Sigma|_s$ then t is a *ground Σ-term*. □

The values of $T_\Sigma(X)$ are "fully-typed" terms formed using the variables in X and the operation names in Σ, and the operations of $T_\Sigma(X)$ just build complicated terms from simpler terms. Note that a term $t \in |T_\Sigma(X)|$ need not contain all the variables in X, and that some variables may occur more than once in t. T_Σ is also called the *Σ-word algebra*, and its carriers $|T_\Sigma|$ are sometimes called the *Herbrand universe for Σ*.

Example 1.4.2. Let $\Sigma 1 = \langle S1, \Omega 1 \rangle$ be as defined in Example 1.2.4. Then $T_{\Sigma 1}$ is the $\Sigma 1$-algebra defined by

$$
\begin{aligned}
|T_{\Sigma 1}|_{shape} = \{ & \text{“}box():shape\text{”}, \\
& \text{“}boxify(box():shape):shape\text{”}, \\
& \text{“}boxify(boxify(box():shape):shape):shape\text{”}, \\
& \dots \}, \\
|T_{\Sigma 1}|_{suit} = \{ & \text{“}hearts():suit\text{”}, \\
& \text{“}f(box():shape, hearts():suit):suit\text{”}, \\
& \text{“}f(boxify(box():shape):shape, hearts():suit):suit\text{”}, \\
& \text{“}f(box():shape, f(box():shape, hearts():suit):suit):suit\text{”}, \\
& \dots \}
\end{aligned}
$$

where the operations of $T_{\Sigma 1}$ are the term formation operations

$$
\begin{aligned}
& box_{T_{\Sigma 1}} = \text{“}box():shape\text{”} \in |T_{\Sigma 1}|_{shape}, \\
& hearts_{T_{\Sigma 1}} = \text{“}hearts():suit\text{”} \in |T_{\Sigma 1}|_{suit}, \\
& boxify_{T_{\Sigma 1}} : |T_{\Sigma 1}|_{shape} \to |T_{\Sigma 1}|_{shape} \\
& \qquad = \{ \text{“}box():shape\text{”} \mapsto \text{“}boxify(box():shape):shape\text{”}, \\
& \qquad\quad \text{“}boxify(box():shape):shape\text{”} \mapsto \text{“}boxify(boxify(box():shape):shape):shape\text{”}, \\
& \qquad\quad \dots \},
\end{aligned}
$$

and similarly for $f : shape \times suit \to suit$. □

Notation. Sort decorations (e.g. “$:shape$” in “$box():shape$”) are often unambiguously determined, and they will usually be omitted when this is the case. When $\Omega_{\varepsilon,s} \cap X_s = \varnothing$ for some $s \in S$, then variables of sort s cannot be confused with constants (0-ary operations) of sort s and so we will usually drop the parentheses “()” in the latter. We will omit quotation marks whenever it is clear from the context that we are dealing with terms. Finally, in examples we will use infix notation for binary operations when convenient. □

Example 1.4.2 (revisited). We repeat Example 1.4.2, making use of these notational conventions.

Let $\Sigma 1 = \langle S1, \Omega 1 \rangle$ be as defined in Example 1.2.4. Then $T_{\Sigma 1}$ is the $\Sigma 1$-algebra defined by

$$
\begin{aligned}
& |T_{\Sigma 1}|_{shape} = \{box, boxify(box), boxify(boxify(box)), \dots\}, \\
& |T_{\Sigma 1}|_{suit} = \{hearts, f(box, hearts), f(boxify(box), hearts), f(box, f(box, hearts)), \dots\}
\end{aligned}
$$

where the operations of $T_{\Sigma 1}$ are the term formation operations

$$
\begin{aligned}
& box_{T_{\Sigma 1}} = box \in |T_{\Sigma 1}|_{shape}, \\
& hearts_{T_{\Sigma 1}} = hearts \in |T_{\Sigma 1}|_{suit}, \\
& boxify_{T_{\Sigma 1}} : |T_{\Sigma 1}|_{shape} \to |T_{\Sigma 1}|_{shape} \\
& \qquad = \{box \mapsto boxify(box), boxify(box) \mapsto boxify(boxify(box)), \dots\},
\end{aligned}
$$

and similarly for $f : shape \times suit \to suit$. □

Example 1.4.3. The notational conventions above will almost always be applicable. They cannot be adopted from the outset (i.e. in Definition 1.4.1) because of the relatively rare examples where confusion can arise. For example, let $\Sigma 2 = \langle S2, \Omega 2 \rangle$

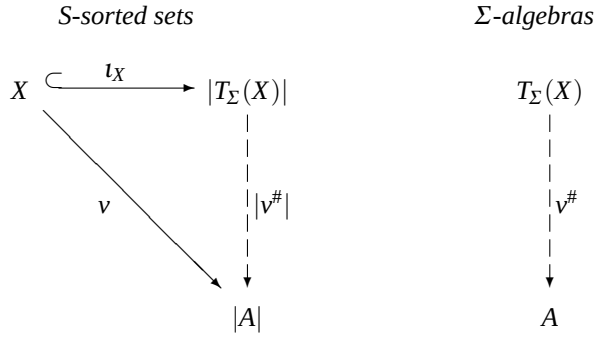
be the signature with sorts s, s', t and operations $a{:}\,s$, $a{:}\,s'$, $f{:}\,s \to t$ and $f{:}\,s' \to t$ (no mistakes here, repetition of names is intented).

According to the definition, $|T_{\Sigma 2}|_t = \{\text{“}f(a():s):t\text{”}, \text{“}f(a():s'):t\text{”}\}$. If all sort decorations were omitted then both of the terms in this set would become “$f(a())$” and so $|T_{\Sigma 2}|_t$ would have just this single element. The “outer” decoration can be omitted but the “inner” decoration is required, thus e.g. “$f(a():s)$”.

Similarly, if X is an $S2$-sorted set of variables such that $a \in X_s$, then “$f(a():s)$” and “$f(a{:}\,s)$” are different terms in $|T_{\Sigma 2}(X)|_t$, so the convention of writing “$a():s$” as “$a{:}\,s$” cannot be used.

Since the definitions permit variables and operation names like $f(a():s)$ and even “ or , or (), the custom of writing terms as sequences of symbols without explicit separators can cause confusion. Luckily, such names never arise in practice and so for the purposes of this book this problem can safely be forgotten. □

Fact 1.4.4. *For any Σ-algebra A and S-sorted function $v{:}\,X \to |A|$ there is exactly one Σ-homomorphism $v^\#{:}\,T_\Sigma(X) \to A$ that extends v, i.e. such that $v^\#_s(\iota_X(x)) = v_s(x)$ for all $s \in S$, $x \in X_s$, where $\iota_X{:}\,X \to |T_\Sigma(X)|$ is the embedding that maps each variable in X to its corresponding term.*



□

The existence and uniqueness of $v^\#$ follow easily from the requirement that $v^\#$ extends v (this fixes the value of $v^\#$ for any variable as a term in $|T_\Sigma(X)|$) and that $v^\#$ is a Σ-homomorphism (this determines the value of $v^\#$ for any term $f(t_1,\ldots,t_n) \in |T_\Sigma(X)|$ as a function of the values of $v^\#$ for its immediate subterms $t_1,\ldots,t_n \in |T_\Sigma(X)|$). The homomorphism which results is the function which evaluates Σ-terms based on the assignment of values in A to variables in X given by v.

Definition 1.4.5 (Term evaluation). Let A be a Σ-algebra A and let $v{:}\,X \to |A|$ be an S-sorted function. By Fact 1.4.4 there is a unique Σ-homomorphism $v^\#{:}\,T_\Sigma(X) \to A$ that extends v. Let $s \in S$ and let $t \in |T_\Sigma(X)|_s$ be a Σ-term of sort s; the *value of t in A under the valuation v* is $v^\#(t) \in |A|_s$. When $t \in |T_\Sigma|_s$ the value of t does not depend on v; then the *value of t in A* is $\varnothing^\#(t)$ where $\varnothing{:}\,\varnothing \to |A|$ is the empty function. To make the algebra explicit, we write $t_A(v)$ for $v^\#(t)$, and t_A for $t_A(\varnothing)$ when t is ground. □

Exercise 1.4.6. Let $t \in |T_\Sigma(X)|$ be a Σ-term and let A be a Σ-algebra. Show that if $v{:}X \to |A|$ and $v'{:}X \to |A|$ coincide on $FV(t)$, then $t_A(v) = t_A(v')$. This follows from another fact: for any $t \in |T_\Sigma(X)|$, $X \subseteq Y$ (so that $t \in |T_\Sigma(Y)|$) and $v{:}Y \to |A|$, we have $t_A(v) = t_A(\iota;v)$, where $\iota{:}X \hookrightarrow Y$ is the inclusion (and so $\iota;v{:}X \to |A|$). □

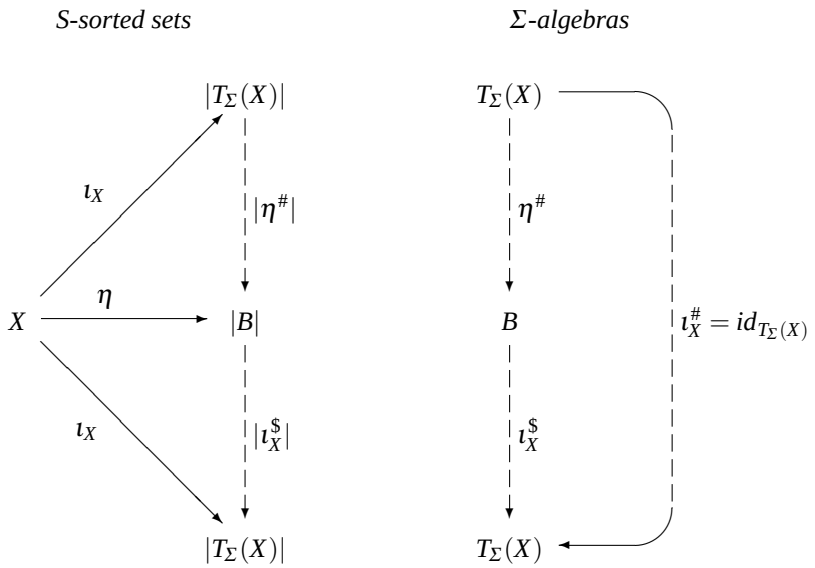
Exercise 1.4.7. Define evaluation of terms in an inductive fashion. Convince yourself that the result is the same as that given by Definition 1.4.5. □

Exercise 1.4.8. Let $h{:}A \to B$ be a Σ-homomorphism, let $v{:}X \to |A|$ be an S-sorted function, and let $t \in |T_\Sigma(X)|$ be a Σ-term. Using Fact 1.4.4, prove that $h(v^\#(t)) = (v;h)^\#(t)$. Compare this with a proof of the same thing using your inductive definition of term evaluation from Exercise 1.4.7. □

Exercise 1.4.9. Functions $\theta{:}X \to |T_\Sigma(Y)|$ are sometimes called *substitutions* (of terms in $T_\Sigma(Y)$ for variables in X). Using Fact 1.4.4, define the Σ-term $t[\theta]$ resulting from applying the substitution θ to a Σ-term $t \in |T_\Sigma(X)|$. Show that $t[\iota_X] = t$ for any $t \in |T_\Sigma(X)|$, where ι_X maps each variable in X to its corresponding term in $|T_\Sigma(X)|$. Define the composition $\theta;\theta'$ of substitutions $\theta{:}X \to |T_\Sigma(Y)|$ and $\theta'{:}Y \to |T_\Sigma(Z)|$, and show that $(t[\theta])[\theta'] = t[\theta;\theta']$ for any Σ-term t and substitutions θ and θ'. □

Notation. Suppose $u \in |T_\Sigma(Y)|_s$ for some sort $s \in S$. Then $[x \mapsto u]$ (when used as a substitution $\{x{:}s\} \cup X \to |T_\Sigma(X \cup Y)|$) is shorthand for the function $\{x{:}s \mapsto u\} \cup \{z \mapsto z \mid z \in X, z \neq x{:}s\}$. For $t \in |T_\Sigma(\{x{:}s\} \cup X)|$, $t[x \mapsto u] \in |T_\Sigma(X \cup Y)|$ thus stands for the term obtained by substituting u for x in t. This notation generalises straightforwardly to $[x_1 \mapsto u_1, \ldots, x_n \mapsto u_n]$ and $t[x_1 \mapsto u_1, \ldots, x_n \mapsto u_n]$ provided $x_1, \ldots, x_n$ are distinct variables. □

Fact 1.4.10. *The property of $T_\Sigma(X)$ in Fact 1.4.4 defines $T_\Sigma(X)$ up to isomorphism: if B is a Σ-algebra and $\eta{:}X \to |B|$ is an S-sorted function such that for any Σ-algebra A and S-sorted function $v{:}X \to |A|$ there is a unique Σ-homomorphism $v^\${:}B \to A$ such that $\eta;|v^\$| = v$ then B is isomorphic to $T_\Sigma(X)$, where $\eta^\#{:}T_\Sigma(X) \to B$ is an isomorphism with inverse $\iota_X^\${:}B \to T_\Sigma(X)$.*



□

Fact 1.4.4 says that the definition of $T_\Sigma(X)$ fixes the definition of the term evaluation function "for free" (see Definition 1.4.5). Fact 1.4.10 says that this property is unique (up to isomorphism) to $T_\Sigma(X)$, so in fact the explicit definition of $T_\Sigma(X)$ is superfluous — it would be enough to define $T_\Sigma(X)$ as "the" (unique up to isomorphism) Σ-algebra for which Definition 1.4.5 makes sense. $T_\Sigma(X)$ is a particular example of a *free object* — see Section 3.5 for more on this topic.

Example 1.4.11. Let $\Sigma 1 = \langle S1, \Omega 1\rangle$ be as defined in Example 1.2.4. Then $T_{\Sigma 1}$ is the $\Sigma 1$-algebra described in Example 1.4.2. Let $T1$ be the $\Sigma 1$-algebra defined by

$$|T1|_{shape} = \{box, box\,boxify, box\,boxify\,boxify, \ldots\},$$
$$|T1|_{suit} = \{hearts, box\,hearts\,f, box\,boxify\,hearts\,f, box\,box\,hearts\,f\,f, \ldots\}$$

where the operations of $T1$ are the postfix term formation operations

$$box_{T1} = box \in |T1|_{shape},$$
$$hearts_{T1} = hearts \in |T1|_{suit},$$
$$boxify_{T1}: |T1|_{shape} \to |T1|_{shape} = \{box \mapsto box\,boxify, box\,boxify \mapsto box\,boxify\,boxify, \ldots\},$$

and similarly for $f: shape \times suit \to suit$. Then $T1$ satisfies the property of $T_{\Sigma 1}$ in Fact 1.4.4 (the fact that $X = \varnothing$ here makes this easy to check — there is only one function $v: \varnothing \to |A1|$ for any $\Sigma 1$-algebra $A1$), so by Fact 1.4.10 (where $\eta: \varnothing \to |T1|$ is the empty function) $T1$ is isomorphic to $T_{\Sigma 1}$. The isomorphism $\varnothing^\#: T_{\Sigma 1} \to T1$ converts a $\Sigma 1$-term to its postfix form. □

Exercise 1.4.12. Prove Facts 1.4.4 and 1.4.10. □

Exercise 1.4.13. Let A be a Σ-algebra and let $\varnothing: \varnothing \to |A|$ be the empty function. Show that A is reachable iff the unique homomorphism $\varnothing^\# : T_\Sigma \to A$ is surjective, i.e., iff every element in $|A|$ is the value of a ground Σ-term. □

Exercise 1.4.14. Show that T_Σ is reachable. Put this fact together with previous results to show that a Σ-algebra is reachable iff it is isomorphic to a quotient of T_Σ, and that there is a one-to-one correspondence between isomorphism classes of reachable Σ-algebras and congruences on T_Σ. □

Exercise 1.4.15. Let G be a context-free grammar over an alphabet T of terminal symbols. Consider the signature $\Sigma_G = \langle S_G, \Omega_G \rangle$, where S_G is the set of non-terminal symbols of G and each production $X \to Y_1 \ldots Y_n$ in G corresponds to an operation in Ω_G with result sort X and arity given by the sequence of non-terminal symbols in $Y_1 \ldots Y_n$. The Σ_G-algebra A_G has carriers $|A_G|_X = T^*$ for all $X \in S_G$, and for any $p: X_1 \times \cdots \times X_n \to X$ in Σ_G and $a_1, \ldots, a_n \in T^*$, $p_{A_G}(a_1, \ldots, a_n)$ is the sequence obtained by substituting a_j for the j^{th} non-terminal symbol on the right-hand side of the production associated with p. Prove the following:

1. For any $X \in S_G$, the carrier of sort X in the reachable subalgebra of A_G is the set of sequences generated from the non-terminal X in G.
2. The algebra T_{Σ_G} is isomorphic to the algebra of parse trees of G.
3. The grammar G is unambiguous iff the reachable subalgebra of A_G is isomorphic to T_{Σ_G}. □

1.5 Changing signatures

A signature morphism defines a mapping from the sort and operation names in one signature to those in another signature, in such a way that the arity and result sort of operations are respected. (This requirement is analogous to the requirement that homomorphisms respect the behaviour of the operations.) Signature morphisms will be used extensively in later chapters to mediate constructions involving multiple signatures. The crucial point that makes these constructions work is that a signature morphism from Σ to Σ' induces translations of syntax (terms — later, also logical formulae) and semantics (algebras and homomorphisms) between Σ and Σ'.

Two kinds of signature morphisms are introduced in this section. Only the first kind will be used in the rest of the book. The second kind, *derived signature morphisms*, are introduced mainly as an example of one way in which a basic definition could be modified. Such a modification would not affect later definitions and results, since these depend only on the induced translations of terms, algebras and homomorphisms.

1.5.1 Signature morphisms

Definition 1.5.1 (Signature morphism). Let $\Sigma = \langle S, \Omega \rangle$ and $\Sigma' = \langle S', \Omega' \rangle$ be signatures. A *signature morphism* $\sigma: \Sigma \to \Sigma'$ is a pair $\sigma = \langle \sigma_{sorts}, \sigma_{ops} \rangle$ where $\sigma_{sorts}: S \to$

S' and σ_{ops} is a family of functions respecting the arities and result sorts of operation names in Σ, that is $\sigma_{ops} = \langle \sigma_{w,s} : \Omega_{w,s} \to \Omega'_{\sigma^*_{sorts}(w),\sigma_{sorts}(s)} \rangle_{w \in S^*, s \in S}$ (where for $w = s_1 \ldots s_n \in S^*$, $\sigma^*_{sorts}(w) = \sigma_{sorts}(s_1) \ldots \sigma_{sorts}(s_n)$). A signature morphism $\sigma : \Sigma \to \Sigma'$ is a *signature inclusion* $\sigma : \Sigma \hookrightarrow \Sigma'$ if σ_{sorts} is an inclusion and $\sigma_{w,s}$ is an inclusion for all $w \in S^*, s \in S$. □

Signature morphisms as defined above will be referred to as *algebraic* signature morphisms when it is necessary to distinguish them from other kinds of signature morphisms to be introduced later. Note that σ_{sorts} and (the functions constituting) σ_{ops} are not required to be either surjective or injective.

Notation. When $\sigma : \Sigma \to \Sigma'$, both σ_{sorts} and σ_{ops} (and its components $\sigma_{w,s}$ for all $w \in S^*, s \in S$) will be denoted by σ. □

Example 1.5.2. Let $\Sigma = \langle S, \Omega \rangle$ be the signature

sorts *polygon*, *figure*, *trump*
ops *square* : *polygon*
boxify : *polygon* → *polygon*
boxify : *polygon* → *figure*
h : *figure* × *trump* → *trump*

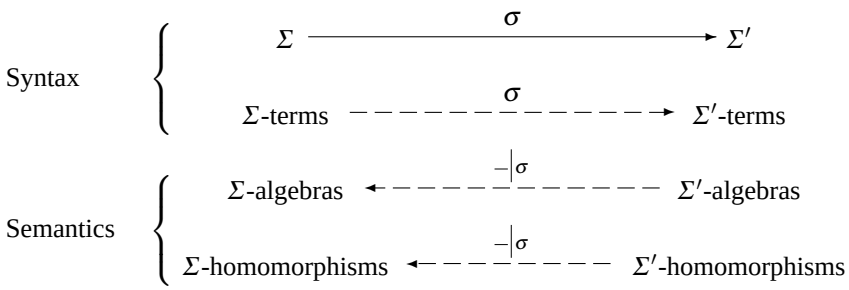
Let $\Sigma 1 = \langle S1, \Omega 1 \rangle$ be the signature defined in Example 1.2.4.
Define $\sigma_{sorts} : S \to S1$ and $\sigma_{ops} = \langle \sigma_{w,s} : \Omega_{w,s} \to \Omega 1_{\sigma^*_{sorts}(w),\sigma_{sorts}(s)} \rangle_{w \in S^*, s \in S}$ by

$\sigma_{sorts} = \{polygon \mapsto shape, figure \mapsto shape, trump \mapsto suit\}$,
$\sigma_{\varepsilon,polygon} = \{square \mapsto box\}$, $\sigma_{polygon,polygon} = \{boxify \mapsto boxify\}$,
$\sigma_{polygon,figure} = \{boxify \mapsto boxify\}$,
$\sigma_{figure\,trump,trump} = \{h \mapsto f\}$,

and $\sigma_{w,s} = \varnothing$ for all other $w \in S^*, s \in S$. Then $\sigma : \Sigma \to \Sigma 1$ is a signature morphism. □

Exercise 1.5.3. Let $\sigma : \Sigma \to \Sigma'$ and $\sigma' : \Sigma' \to \Sigma''$ be signature morphisms. Let $(\sigma;\sigma')_{sorts} = \sigma_{sorts};\sigma'_{sorts}$ and $(\sigma;\sigma')_{ops} = \sigma_{ops};\sigma'_{ops}$ (or rather, to be more precise: $(\sigma;\sigma')_{w,s} = \sigma_{w,s};\sigma'_{\sigma^*_{sorts}(w),\sigma_{sorts}(s)}$ for $w \in S^*, s \in S$). Show that this defines a signature morphism $\sigma;\sigma' : \Sigma \to \Sigma''$. □

In the rest of this section, let $\sigma : \Sigma \to \Sigma'$ be a signature morphism, where $\Sigma = \langle S, \Omega \rangle$ and $\Sigma' = \langle S', \Omega' \rangle$. As will be defined below, any such signature morphism gives rise to a translation of Σ-terms to Σ'-terms, and of Σ'-algebras and homomorphisms to Σ-algebras and homomorphisms. Note that the direction of translation of algebras and homomorphisms is "backwards" with respect to the direction of the signature morphism, as the following figure indicates.



Definition 1.5.4 (Reduct algebra). Let A' be a Σ'-algebra. The *σ-reduct of A'* is the Σ-algebra $A'|_\sigma$ defined as follows:

- $|A'|_\sigma|_s = |A'|_{\sigma(s)}$ for all $s \in S$; and
- for all $f : s_1 \times \cdots \times s_n \to s$ in Σ,

$$f_{A'|_\sigma} : |A'|_\sigma|_{s_1} \times \cdots \times |A'|_\sigma|_{s_n} \to |A'|_\sigma|_s = \sigma(f)_{A'} : |A'|_{\sigma(s_1)} \times \cdots \times |A'|_{\sigma(s_n)} \to |A'|_{\sigma(s)}.$$

□

If Σ is a subsignature of Σ', $\sigma : \Sigma \hookrightarrow \Sigma'$ is the signature inclusion, and A' is a Σ'-algebra, then $A'|_\sigma$ is a Σ-algebra which is just A' with some carriers and/or operations removed.

Notation. We sometimes write $A'|_\Sigma$ for $A'|_\sigma$ when $\sigma : \Sigma \to \Sigma'$ is obvious, such as when σ is a signature inclusion. □

Example 1.5.5. Let $\sigma : \Sigma \to \Sigma 1$ be the signature morphism defined in Example 1.5.2 and let $A1$ be the $\Sigma 1$-algebra defined in Example 1.2.4. Then $A1|_\sigma$ is the Σ-algebra such that

$$\begin{aligned}
&|A1|_\sigma|_{polygon} = |A1|_\sigma|_{figure} = \{\square, \triangle\} = |A1|_{shape},\\
&|A1|_\sigma|_{trump} = \{\clubsuit, \heartsuit, \spadesuit\} = |A1|_{suit},\\
&square_{A1|_\sigma} = \square = box_{A1},\\
&boxify_{A1|_\sigma} : |A1|_\sigma|_{polygon} \to |A1|_\sigma|_{polygon} = \{\square \mapsto \square, \triangle \mapsto \square\}\\
&\qquad\qquad = boxify_{A1} : |A1|_{shape} \to |A1|_{shape},\\
&boxify_{A1|_\sigma} : |A1|_\sigma|_{polygon} \to |A1|_\sigma|_{figure} = \{\square \mapsto \square, \triangle \mapsto \square\}\\
&\qquad\qquad = boxify_{A1} : |A1|_{shape} \to |A1|_{shape},\\
&h_{A1|_\sigma} : |A1|_\sigma|_{figure} \times |A1|_\sigma|_{trump} \to |A1|_\sigma|_{trump} = \{\langle\square, \clubsuit\rangle \mapsto \clubsuit, \langle\square, \heartsuit\rangle \mapsto \spadesuit, \ldots\}\\
&\qquad\qquad = f_{A1} : |A1|_{shape} \times |A1|_{suit} \to |A1|_{suit}.
\end{aligned}$$

□

Exercise 1.5.6. A Σ-algebra A can be regarded as a function mapping the names in Σ to their interpretations; the σ-reduct of A is then the composition $\sigma;A$. Spell out the details. □

Exercise 1.5.7. Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism that is surjective on sort names, and let A' be a Σ'-algebra. Show that if $A'|_\sigma$ is reachable then A' is reachable. Give counterexamples showing that the opposite implication does not hold, and that the implication itself does not hold if some sort names in Σ' are not in the image of Σ under σ. □

Definition 1.5.8 (Reduct homomorphism). Let $h' : A' \to B'$ be a Σ'-homomorphism. The *σ-reduct of h'* is the S-sorted function $h'|_\sigma : |A'|_\sigma| \to |B'|_\sigma|$ such that $(h'|_\sigma)_s = h'_{\sigma(s)}$ for all $s \in S$. (**Exercise:** Show that $h'|_\sigma : A'|_\sigma \to B'|_\sigma$ is a Σ-homomorphism.) □

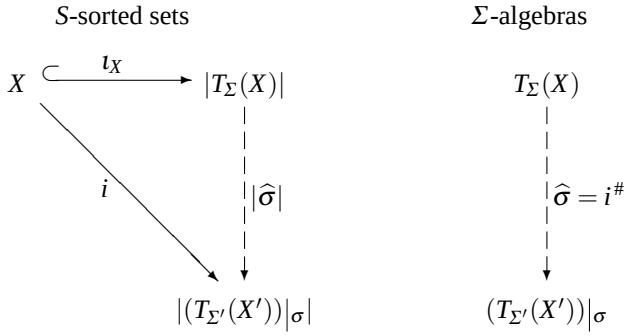
Exercise 1.5.9. Define the σ-reduct $\equiv'|_\sigma$ of a Σ'-congruence $\equiv'$ on a Σ'-algebra A', and prove that it is a Σ-congruence on $A'|_\sigma$. Show that σ-reduct distributes over quotient, i.e. $(A'/\equiv')|_\sigma = (A'|_\sigma)/(\equiv'|_\sigma)$ for all Σ'-algebras A' and Σ'-congruences $\equiv'$ on A'. □

The following definition of the translation of terms along a signature morphism $\sigma : \Sigma \to \Sigma'$ may look somewhat daunting, but its simple upshot is to translate each term $t \in |T_\Sigma(X)|$ to the Σ'-term obtained by replacing each operation name from Σ by its image under σ. Some care must be taken in the treatment of variables: since variables for different sorts are not required to be distinct, to make sure they are not inadvertently identified by the translation, for each sort s' in Σ' we have to take a disjoint union of the sets of variables of sorts mapped to s'.

Definition 1.5.10 (Term translation). Let X be an S-sorted set of variables. Define $X' = \langle X'_{s'} \rangle_{s' \in S'}$ to be the S'-sorted set such that

$$X'_{s'} = \biguplus_{\sigma(s)=s'} X_s \qquad \text{for each } s' \in S'.$$

Then $(T_{\Sigma'}(X'))|_\sigma$ is a Σ-algebra. Let $i : X \to |(T_{\Sigma'}(X'))|_\sigma|$ be the obvious embedding (if not for the disjoint union in the definition of X' and explicit decoration of variables with sorts in terms, i would coincide with ι_X which maps each variable to its corresponding term). Then by Fact 1.4.4 there is a unique Σ-homomorphism $\widehat{\sigma} : T_\Sigma(X) \to (T_{\Sigma'}(X'))|_\sigma$ extending i:



The *translation of a Σ-term $t \in |T_\Sigma(X)|$ by σ* is the Σ'-term $\widehat{\sigma}(t) \in |T_{\Sigma'}(X')|$. To keep the notation simple, we will write just $\sigma(t)$ for $\widehat{\sigma}(t)$. □

Example 1.5.11. Let $\sigma : \Sigma \to \Sigma 1$ be the signature morphism defined in Example 1.5.2, where $\Sigma = \langle S, \Omega \rangle$ and $\Sigma 1 = \langle S1, \Omega 1 \rangle$. Let X be the S-sorted set of variables $x : \mathit{polygon}, x : \mathit{figure}, y : \mathit{figure}, z : \mathit{trump}$. The $S1$-sorted set of variables X' in Definition 1.5.10 is then $x : \mathit{shape}, x' : \mathit{shape}, y : \mathit{shape}, z : \mathit{suit}$, and

$$\sigma(h(boxify(x{:}polygon), h(x{:}figure, z))) = f(boxify(x), f(x', z)),$$

$$\sigma(h(x{:}figure, h(boxify(boxify(square)), z))) = f(x', f(boxify(boxify(box)), z)),$$

and so on. □

Exercise 1.5.12. Let $t \in |T_\Sigma|$ be a ground Σ-term and let A' be a Σ'-algebra. Show that the value of t is invariant under change of signature, i.e. $\sigma(t)_{A'} = t_{A'|_\sigma}$.

Formulate and prove a more general version of this result in which t may contain variables. □

1.5.2 Derived signature morphisms

A derived signature morphism from Σ to Σ' is like an algebraic signature morphism from Σ to Σ' except that operation names in Σ are mapped to *terms* over Σ'. This allows operation names in Σ to be mapped to combinations of operations in Σ', and also handles the case where the order of arguments of the corresponding operations in Σ and Σ' are different.

Definition 1.5.13 (Derived signature). Let $\Sigma = \langle S, \Omega \rangle$ be a signature. For any sequence $s_1 \ldots s_n \in S^*$, let $I_{s_1 \ldots s_n}$ be the S-sorted set $\boxed{1}{:}s_1, \ldots, \boxed{n}{:}s_n$. The *derived signature of* Σ is the signature $\Sigma^{der} = \langle S, \Omega^{der} \rangle$ where for each $s_1 \ldots s_n \in S^*$ and $s \in S$, $\Omega^{der}_{s_1 \ldots s_n, s} = |T_\Sigma(I_{s_1 \ldots s_n})|_s$. □

In the derived signature of Σ, a Σ-term t of sort s with variables $I_{s_1 \ldots s_n}$ represents an operation $t{:}\, s_1 \times \cdots \times s_n \to s$. The variable $\boxed{i}{:}s_i$ in $I_{s_1 \ldots s_n}$ thus stands for the ith argument of t. Note that a "bare" variable $\boxed{i} \in |T_\Sigma(I_{s_1 \ldots s_n})|_{s_i}$ is an operation $i{:}\, s_1 \times \cdots \times s_n \to s_i$ in Σ^{der}, corresponding to a projection function.

Definition 1.5.14 (Derived signature morphism). Let Σ and Σ' be signatures. A *derived signature morphism* $\delta{:}\, \Sigma \to \Sigma'$ is an algebraic signature morphism $\delta{:}\, \Sigma \to (\Sigma')^{der}$. □

Definition 1.5.15 (Derived algebra). Let $\Sigma = \langle S, \Omega \rangle$ be a signature, and let A be a Σ-algebra. The *derived algebra of* A is the Σ^{der}-algebra A^{der} defined as follows:

- $|A^{der}| = |A|$; and
- for each $t{:}\, s_1 \times \cdots \times s_n \to s$ in Σ^{der} and $a_1 \in |A^{der}|_{s_1}, \ldots, a_n \in |A^{der}|_{s_n}$, $t_{A^{der}}(a_1, \ldots, a_n) = t_A(v) \in |A^{der}|_s$ where v is the S-sorted function $\{(\boxed{1}{:}s_1) \mapsto a_1, \ldots, (\boxed{n}{:}s_n) \mapsto a_n\}$. □

In the rest of this section, let $\delta{:}\, \Sigma \to \Sigma'$ be a derived signature morphism. The following corresponds to Definition 1.5.4 for algebraic signature morphisms; later exercises correspond to Definitions 1.5.8 and 1.5.10 and related results.

Definition 1.5.16 (Reduct algebra w.r.t. a derived signature morphism). Let A' be a Σ'-algebra. The *δ-reduct of* A' is the Σ-algebra $A'|_\delta$ defined as follows:

- $|A'|_\delta|_s = |A'|_{\delta(s)}$ for all $s \in S$; and
- for all $f: s_1 \times \cdots \times s_n \to s$ in Σ, $f_{A'|_\delta}: |A'|_\delta|_{s_1} \times \cdots \times |A'|_\delta|_{s_n} \to |A'|_\delta|_s = \delta(f)_{(A')^{der}}$.

Equivalently, $A'|_\delta$ is the Σ-algebra $(A')^{der}|_\delta$, viewing δ as the algebraic signature morphism $\delta: \Sigma \to (\Sigma')^{der}$. □

Exercise 1.5.17 (Reduct homomorphism w.r.t. a derived signature morphism). What is the δ-reduct $h'|_\delta$ of a Σ'-homomorphism $h': A' \to B'$? Prove that $h'|_\delta: A'|_\delta \to B'|_\delta$ is a Σ-homomorphism. □

Exercise 1.5.18 (Term translation w.r.t. a derived signature morphism). Let $t \in |T_\Sigma(X)|$ be a Σ-term, where X is an S-sorted set of variables. Define $\delta(t)$, the translation of t by δ (the result should be a Σ'-term). □

Example 1.5.19. Let $\Sigma = \langle S, \Omega \rangle$ be the signature defined in Example 1.5.2, and let $\Sigma 1 = \langle S1, \Omega 1 \rangle$ be the signature defined in Example 1.2.4. Let $\delta: \Sigma \to \Sigma 1$ be the derived signature morphism defined by

$$\begin{aligned}
&\delta_{sorts} = \{polygon \mapsto shape, figure \mapsto shape, trump \mapsto suit\},\\
&\delta_{\varepsilon,polygon} = \{square \mapsto boxify(box)\},\\
&\delta_{polygon,polygon} = \{boxify \mapsto \boxed{1}\!:shape\},\\
&\delta_{polygon,figure} = \{boxify \mapsto boxify(boxify(\boxed{1}\!:shape))\},\\
&\delta_{figure\,trump,trump} = \{h \mapsto f(boxify(\boxed{1}\!:shape), f(\boxed{1}\!:shape, \boxed{2}\!:suit))\},
\end{aligned}$$

and $\delta_{w,s} = \varnothing$ for all other $w \in S^*, s \in S$.

Let $A1$ be the $\Sigma 1$-algebra defined in Example 1.2.4. Then $A1|_\delta$ is the Σ-algebra such that

$$\begin{aligned}
&|A1|_\delta|_{polygon} = |A1|_\delta|_{figure} = \{\square, \triangle\},\\
&|A1|_\delta|_{trump} = \{\clubsuit, \heartsuit, \spadesuit\},\\
&square_{A1|_\delta} = \square,\\
&boxify_{A1|_\delta}: |A1|_\delta|_{polygon} \to |A1|_\delta|_{polygon} = \{\square \mapsto \square, \triangle \mapsto \triangle\}\\
&boxify_{A1|_\delta}: |A1|_\delta|_{polygon} \to |A1|_\delta|_{figure} = \{\square \mapsto \square, \triangle \mapsto \square\},
\end{aligned}$$

and $h_{A1|_\delta}: |A1|_\delta|_{figure} \times |A1|_\delta|_{trump} \to |A1|_\delta|_{trump}$ is defined by the following table:

$h_{A1\|_\delta}$	$\clubsuit$	$\heartsuit$	$\spadesuit$
$\square$	$\clubsuit$	$\heartsuit$	$\spadesuit$
$\triangle$	$\spadesuit$	$\heartsuit$	$\heartsuit$

Let X be the S-sorted set of variables x:*polygon*, x:*figure*, y:*figure*, z:*trump*. A correct solution to Exercise 1.5.18 would translate $h(boxify(x\!:polygon), h(x\!:figure, z))$ (a Σ-term with variables X) to

$$f(boxify(\underbrace{boxify(boxify(x))}_{=\delta(boxify(x:polygon))}), f(\underbrace{boxify(boxify(x))}_{=\delta(boxify(x:polygon))}, \underbrace{f(boxify(x'), f(x', z))}_{=\delta(h(x:figure,z))})).$$

□

Exercise 1.5.20. Repeat Exercise 1.5.12 for the case of derived signature morphisms. □

Exercise 1.5.21. A more complex definition of derived signature morphism $\delta : \Sigma \to \Sigma'$ would allow a sort name s in Σ to be mapped to a *Cartesian product* $s'_1 \times \cdots \times s'_n$ of sorts $s'_1, \ldots, s'_n$ in Σ'. Give versions of the above definitions which permit this. □

Exercise 1.5.22. Another variation on the definition of derived signature morphism would permit operation names in Σ to be mapped to recursively defined functions in terms of the operation names in Σ'. Give versions of the above definitions which would allow this. (HINT: Look at a book like [Sch86] before attempting this exercise.) □

1.6 Bibliographical remarks

This chapter presents the basic notions of universal algebra that are required in the sequel. There is a vast literature on universal algebra as a branch of mathematics, and the concepts and results we need here are a tiny fraction of this. Applications of universal algebra in computer science are widespread, going back at least to [BL69].

For much more on universal algebra see e.g. [Grä79] or [Coh65] but note that both of these handle only the single-sorted case. A presentation of some of this material for a computer scientist audience is [Wec92], see also [MT92] where applications to some topics in computer science other than the ones covered in this book are also indicated.

The style of presentation here is relaxed but it might still be too dense for some readers, who might prefer the gentler style, with proofs of many of the results which we omit here, in [GTW76], [EM85], [MG85] or [LEW96].

The generalisation from single-sorted to many-sorted algebras originates with [Hig63]. First applications to computer science came later [Mai72], becoming prominent with [GTW76]. The generalisation is straightforward from a purely mathematical standpoint, but there are a few subtle issues that will surface in later chapters. For instance, we admit empty carrier sets in Definition 1.2.2, unlike most logic books and, for instance, [BT87] and [Mos04]. Admitting empty carrier sets requires more care in the presentation of rules for reasoning, see Exercise 2.4.10 below, but it also makes some results smoother, see Exercise 2.5.18.

There are different definitions of many-sorted signature in the literature. The one here is quite general, allowing overloading of operation names etc., and originates with [GTWW73] and [Gog74]. In some early papers, signatures are called "operator domains". Definitions that do not permit overloading are used in [EM85] and [Wir90], but as remarked after Definition 1.2.1, these definitions are equivalent if each operation name is taken to be tagged with its arity and result sort.

Signature morphisms emerged around 1978 in the context of early work on the semantics of parameterised specifications in the style of Definition 6.3.5 below, see

[Ehr78] and [GB78]; Definition 1.5.1 is from the latter. Various variants and restrictions on this notion have been used in the meantime. One possible simplifying assumption is to restrict attention to injective signature morphisms as in [BHK90], or to bijective signature morphisms, which are sometimes referred to as "renamings". The notion of reduct, but only with respect to a signature inclusion, arises in universal algebra. The generalisation from signature morphisms to derived signature morphisms originates in [GTW76], and is related to the even more general notion of (theory) interpretation in logic [End72]. Since the 1970s, derived signature morphisms have made only sporadic appearances in the algebraic specification literature, see for instance [SB83] and [HLST00].

Chapter 2
Simple equational specifications

A specification is an unambiguous description of a signature Σ and a class of Σ-algebras. Because we model programs as algebras, a specification amounts to a characterisation of a class of programs. Each of these programs is regarded as a correct realisation of the specification.

Given a signature Σ (which, if finite, may be presented by simply listing its sort names and its operation names with their arities and result sorts), there are two basic techniques that may be used for describing a class of Σ-algebras. The first is to simply give a list of all the algebras in the class. Unfortunately, we are almost always interested in *infinite* classes of algebras, where this technique is useless. The second is to describe the functional behaviour of the algebras in the class by listing the properties (axioms) they are to satisfy. This is the fundamental specification technique used in work on algebraic specification and the one that will be studied in this chapter. The simplest and most common case is the one in which properties are expressed in the form of universally quantified equations; in most of this chapter, we restrict attention to this case. Section 2.7 indicates other forms of axioms that may be of use, along with some possible variations on the definitions of Chapter 1, and further possibilities will be discussed in Chapter 4. Since most of the results in this chapter are fairly standard and proofs are readily available in the literature, most proofs are left as exercises for the reader.

Chapters 5 and 8 will cover additional techniques for describing classes of algebras. All of these involve taking a class of algebras and performing a simple operation to obtain another class of algebras, often over a different signature. Using such methods, complex specifications of classes of complex algebras may be built from small and easily understood units.

2.1 Equations

Any given signature characterises the class of algebras over that signature. Although this fixes the names of sorts and operations, it is an exceedingly limited form of de-

scription since each such class contains a wide diversity of different algebras. Any two algebras taken from such a class may have carrier sets of different cardinalities and containing different elements; even if both algebras happen to have "matching" carrier sets, the results produced by applying operations may differ. For most applications it is necessary to focus on a subclass of algebras, obtained by imposing *axioms* which serve as constraints on the permitted behaviour of operations. One particularly simple form of axioms are equations, which constrain behaviour by asserting that the value of two given terms are *the same*. Equations have limited expressive power, but this disadvantage is to some extent balanced by the simplicity and convenience of reasoning in equational logic (see Sections 2.4 and 2.6).

Variables in equations will be taken from a fixed but arbitrary infinite set $\mathscr{X}$. We require $\mathscr{X}$ to be closed under finite disjoint union: if $\langle X_i \rangle_{i \in I}$ is finite and $X_i \subseteq \mathscr{X}$ for all $i \in I$, then $\biguplus \langle X_i \rangle_{i \in I} \subseteq \mathscr{X}$. We use variable names like x, y, z in examples, and so we assume that these are all in $\mathscr{X}$. Throughout this section, let $\Sigma = \langle S, \Omega \rangle$ be a signature.

Definition 2.1.1 (Equation). A Σ-*equation* $\forall X \bullet t = t'$ consists of:

- a finite S-sorted set X (of variables), such that $X_s \subseteq \mathscr{X}$ for all $s \in S$; and
- two Σ-terms $t, t' \in |T_\Sigma(X)|_s$ for some sort $s \in S$.

A Σ-equation $\forall \varnothing \bullet t = t'$ is called a *ground (Σ-)equation*. □

Notation. The explicit quantification over X in a Σ-equation $\forall X \bullet t = t'$ is essential, as will become clear in Section 2.4. In spite of this fact, it is common in practice to leave quantification implicit, writing $t = t'$ in place of $\forall FV(t) \cup FV(t') \bullet t = t'$, and we will follow this convention in examples when no confusion is possible. □

Definition 2.1.2 (Satisfaction). A Σ-algebra A *satisfies* (or, *is a model of*) a Σ-equation $\forall X \bullet t = t'$, written $A \models_\Sigma \forall X \bullet t = t'$, if for every ($S$-sorted) function $v : X \to |A|$, $t_A(v) = t'_A(v)$.

A satisfies (or, is a model of) a set Φ of Σ-equations, written $A \models_\Sigma \Phi$, if $A \models_\Sigma \varphi$ for every equation $\varphi \in \Phi$. A class $\mathscr{A}$ of Σ-algebras satisfies a Σ-equation φ, written $\mathscr{A} \models_\Sigma \varphi$, if $A \models_\Sigma \varphi$ for every $A \in \mathscr{A}$. Finally, a class $\mathscr{A}$ of Σ-algebras satisfies a set Φ of Σ-equations, written $\mathscr{A} \models_\Sigma \Phi$, if $A \models_\Sigma \Phi$ for every $A \in \mathscr{A}$ (equivalently, if $\mathscr{A} \models_\Sigma \varphi$ for every $\varphi \in \Phi$, i.e. $A \models_\Sigma \varphi$ for every $A \in \mathscr{A}$ and $\varphi \in \Phi$). □

The definition of satisfaction provides the syntax of equations with the obvious semantics: an algebra A satisfies an equation $\forall X \bullet t = t'$ if for any given assignment of values in $|A|$ to the variables in X, the terms t and t' evaluate in A to the same value.

Notation. We sometimes write $\models$ in place of $\models_\Sigma$ when Σ is obvious. □

Exercise 2.1.3. Recall $\Sigma 1$ and $A1$ from Example 1.2.4. Give some $\Sigma 1$-equations (both ground and non-ground) that are satisfied by $A1$. Give some $\Sigma 1$-equations (both ground and non-ground) that are *not* satisfied by $A1$. □

Exercise 2.1.4. If $\forall X \bullet t = t'$ is a Σ-equation and $X \subseteq X'$ (and $X'_s \subseteq \mathscr{X}$ for all $s \in S$), it follows from Definition 2.1.1 that $\forall X' \bullet t = t'$ is also a Σ-equation. Show that $A \models_\Sigma \forall X \bullet t = t'$ implies that $A \models_\Sigma \forall X' \bullet t = t'$. Give a counterexample showing that the converse does *not* hold. (HINT: Consider $X_s = \varnothing$ and $|A|_s = \varnothing$ for some $s \in S$.) Show that it *does* hold if Σ has only one sort. □

Exercise 2.1.5. Show that surjective Σ-homomorphisms preserve satisfaction of Σ-equations: if $h: A \to B$ is a surjective Σ-homomorphism then $A \models_\Sigma \varphi$ implies $B \models_\Sigma \varphi$ for any Σ-equation φ. Show that injective Σ-homomorphisms reflect satisfaction of Σ-equations: if $h: A \to B$ is an injective Σ-homomorphism then $B \models_\Sigma \varphi$ implies $A \models_\Sigma \varphi$ for any Σ-equation φ. Conclude that Σ-isomorphisms preserve and reflect satisfaction of Σ-equations. □

Exercise 2.1.6. Give an alternative definition of $A \models_\Sigma \forall X \bullet t = t'$ via the satisfaction of $t = t'$ viewed as a ground equation over an enlarged signature. (HINT: Definition 2.1.2 involves quantification over valuations $v: X \to |A|$. Consider how this might be replaced by quantification over algebras having a signature obtained from Σ by adding a constant for each variable in X.) □

It is worth noting in passing the use of the word "class" above to refer to an arbitrary collection of Σ-algebras. We use this term since the collection of Σ-algebras is too "large" to form a set. Since the set/class distinction is peripheral to our concerns here, we will not belabour it, except to mention that it would be possible to avoid the issue entirely by restricting attention to algebras in which all carrier sets are subsets of some large but fixed universal set of values.

A signature morphism $\sigma: \Sigma \to \Sigma'$ gives rise to a translation of Σ-equations to Σ'-equations. This is essentially a simple matter of applying the translation on terms induced by σ to both sides of the equation.

Definition 2.1.7 (Equation translation). Let $\forall X \bullet t = t'$ be a Σ-equation, and let $\sigma: \Sigma \to \Sigma'$ be a signature morphism. Recall from Definition 1.5.10 that we then have $\sigma(t), \sigma(t') \in |T_{\Sigma'}(X')|$ where

$$X'_{s'} = \biguplus_{\sigma(s)=s'} X_s \qquad \text{for each } s' \in S'.$$

The *translation of* $\forall X \bullet t = t'$ *by* σ is then the Σ'-equation $\sigma(\forall X \bullet t = t') = \forall X' \bullet \sigma(t) = \sigma(t')$. (The fact that $\mathscr{X}$ is closed under finite disjoint union guarantees that this is indeed a Σ'-equation.) □

An important result which brings together some of the main definitions above is the following:

Lemma 2.1.8 (Satisfaction Lemma [BG80]). *If $\sigma: \Sigma \to \Sigma'$ is a signature morphism, φ is a Σ-equation and A' is a Σ'-algebra, then $A' \models_{\Sigma'} \sigma(\varphi)$ iff $A'|_\sigma \models_\Sigma \varphi$.* □

When φ is a ground Σ-equation, it is easy to see that this follows directly from the property established in Exercise 1.5.12. When σ is injective (on both sort and operation names), it seems intuitively clear that the Satisfaction Lemma should hold, since the domain of quantification of variables is unchanged, the only difference between φ and $\sigma(\varphi)$ is the names used for sorts and operations, and the only difference between A' and $A'|_\sigma$ (apart from sort/operation names) is that A' might provide interpretations for sort and operation names which do not appear in $\sigma(\varphi)$ and so cannot affect its satisfaction. When σ is non-injective the Satisfaction Lemma still holds, but this is less intuitively obvious (particularly when σ is non-injective on sort names).

Exercise 2.1.9. Take a signature morphism $\sigma: \Sigma \to \Sigma'$ which is non-injective on sort and operation names, a Σ-equation involving the sort and operation names for which σ is not injective, and a Σ'-algebra, and check that the Satisfaction Lemma holds in this case. □

Exercise 2.1.10. Prove the Satisfaction Lemma, using Exercise 1.5.12. □

Exercise 2.1.11. Define the translation of a Σ-equation by a derived signature morphism $\delta: \Sigma \to \Sigma'$, and convince yourself that the Satisfaction Lemma also holds for this case. □

The Satisfaction Lemma says that the translations of syntax (terms, equations) and semantics (algebras) induced by signature morphisms are coherent with the definition of satisfaction. Said another way, the manner in which satisfaction of equations by algebras varies according to the signature at hand fits exactly with these translations. Further discussion of the property embodied in the Satisfaction Lemma may be found in Section 4.1.

2.2 Flat specifications

A signature together with a set of equations over that signature constitutes a simple form of specification. We refer to these as *flat* (meaning *unstructured*) specifications in order to distinguish them from the *structured* specifications to be introduced in Chapter 5, formed from simpler specifications using specification-building operations. As we shall see later, it is possible in some (but not all) cases to "flatten" a structured specification to yield a flat specification describing the same class of algebras.

Throughout this section, let Σ be a signature.

Definition 2.2.1 (Presentation). A *presentation* (also known as a *flat specification*) is a pair $\langle \Sigma, \Phi \rangle$ where Φ is a set of Σ-equations (called the *axioms* of $\langle \Sigma, \Phi \rangle$). A presentation $\langle \Sigma, \Phi \rangle$ is sometimes referred to as a *Σ-presentation.* □

The term "presentation" is chosen to emphasize the syntactic nature of the concept. The idea is that a presentation *denotes* (or *presents*) a semantic object which is

inconvenient to describe directly. A reasonable objection to the definition above is that it fails to include restrictions to ensure that presentations are truly syntactic objects, namely that Σ and Φ are *finite*, or at least effectively presentable in some other sense (e.g. recursive or recursively enumerable). Although it would be possible to impose such a restriction, we refrain from doing so in order to avoid placing undue emphasis on issues of this kind.

Definition 2.2.2 (Model of a presentation). A *model* of a presentation $\langle \Sigma, \Phi \rangle$ is a Σ-algebra A such that $A \models_\Sigma \Phi$. $Mod[\langle \Sigma, \Phi \rangle]$ is the class of all models of $\langle \Sigma, \Phi \rangle$. □

Taking $\langle \Sigma, \Phi \rangle$ to denote the semantic object $Mod[\langle \Sigma, \Phi \rangle]$ is sometimes called taking its *loose semantics*. The word "loose" here refers to the fact that this is not always (in fact, hardly ever) an isomorphism class of algebras: $A, B \in Mod[\langle \Sigma, \Phi \rangle]$ does *not* imply that $A \cong B$. In Section 2.5 we will consider the so-called *initial semantics* of presentations in which a further constraint is imposed on the models of a presentation, forcing every presentation to denote an isomorphism class of algebras.

Example 2.2.3. Let $\mathrm{BOOL} = \langle \Sigma\mathrm{BOOL}, \Phi\mathrm{BOOL} \rangle$ be the presentation below.[1]

spec BOOL =
sorts *bool*
ops *true*: *bool*
false: *bool*
$\neg\_\_$: *bool* → *bool*
$\_\_ \wedge \_\_$: *bool* × *bool* → *bool*
$\_\_ \Rightarrow \_\_$: *bool* × *bool* → *bool*
$\forall p, q$: *bool*
- $\neg true = false$
- $\neg false = true$
- $p \wedge true = p$
- $p \wedge \neg p = false$
- $p \Rightarrow q = \neg(p \wedge \neg q)$

Define ΣBOOL-algebras $A1$, $A2$ and $A3$ as follows:

[1] Here and in the sequel we follow the notation of CASL and itemize axioms in specifications, marking them with • and introducing universal quantification over the variables only once for the rest of the list of axioms. Note though that it may be important to keep some axioms outside of the scope of quantification over some variables, see Exercise 2.1.4.

$|A1|_{bool} = \{\star\}$
$true_{A1} = \star$
$false_{A1} = \star$

$|A2|_{bool} = \{\clubsuit, \heartsuit, \spadesuit\}$
$true_{A2} = \clubsuit$
$false_{A2} = \heartsuit$

$|A3|_{bool} = \{tt, ff\}$
$true_{A3} = tt$
$false_{A3} = ff$

$\neg_{A1} = \{\star \mapsto \star\}$

$\neg_{A2} = \{\clubsuit \mapsto \heartsuit, \heartsuit \mapsto \clubsuit, \spadesuit \mapsto \spadesuit\}$

$\neg_{A3} = \{tt \mapsto ff, ff \mapsto tt\}$

$\wedge_{A1}$	$\star$
$\star$	$\star$

$\wedge_{A2}$	$\clubsuit$	$\heartsuit$	$\spadesuit$
$\clubsuit$	$\clubsuit$	$\heartsuit$	$\heartsuit$
$\heartsuit$	$\heartsuit$	$\heartsuit$	$\heartsuit$
$\spadesuit$	$\spadesuit$	$\heartsuit$	$\heartsuit$

$\wedge_{A3}$	tt	ff
tt	tt	ff
ff	ff	ff

$\Rightarrow_{A1}$	$\star$
$\star$	$\star$

$\Rightarrow_{A2}$	$\clubsuit$	$\heartsuit$	$\spadesuit$
$\clubsuit$	$\clubsuit$	$\heartsuit$	$\clubsuit$
$\heartsuit$	$\clubsuit$	$\clubsuit$	$\clubsuit$
$\spadesuit$	$\clubsuit$	$\spadesuit$	$\clubsuit$

$\Rightarrow_{A3}$	tt	ff
tt	tt	ff
ff	tt	tt

Each of these algebras is a model of BOOL. (NOTE: Reference will be made to BOOL and to its models $A1$, $A2$ and $A3$ in later sections of this chapter. The name BOOL has been chosen for the same reason as `bool` is used for the type of truth values in programming languages; it is technically a misnomer since this is not a specification of Boolean algebras, see Example 2.2.4 below.)

Exercise. Show that the models defined and in fact all the models of BOOL satisfy $\forall p{:}bool \bullet \neg(p \wedge \neg false) = \neg p$. Define a model of BOOL that does not satisfy $\forall p{:}bool \bullet \neg\neg p = p$. □

Example 2.2.4. Let $\mathrm{BA} = \langle \Sigma\mathrm{BA}, \Phi\mathrm{BA} \rangle$ be the following presentation.

spec $\mathrm{BA} =$ **sorts** *bool*
ops $\mathsf{true}: bool$
$\mathsf{false}: bool$
$\neg\_: bool \to bool$
$\_\vee\_: bool \times bool \to bool$
$\_\wedge\_: bool \times bool \to bool$
$\_\Rightarrow\_: bool \times bool \to bool$
$\forall p, q, r: bool$
- $p \vee (q \vee r) = (p \vee q) \vee r$
- $p \wedge (q \wedge r) = (p \wedge q) \wedge r$
- $p \vee q = q \vee p$
- $p \wedge q = q \wedge p$
- $p \vee (p \wedge q) = p$
- $p \wedge (p \vee q) = p$
- $p \vee (q \wedge r) = (p \wedge q) \vee (p \wedge r)$
- $p \wedge (q \vee r) = (p \vee q) \wedge (p \vee r)$
- $p \vee \neg p = \mathsf{true}$
- $p \wedge \neg p = \mathsf{false}$
- $p \Rightarrow q = \neg p \vee q$

Models of BA are called *Boolean algebras*. One such model is the following two-valued Boolean algebra $\mathbb{B}$:

$|\mathbb{B}|_{bool} = \{tt, ff\}$,
$\mathsf{true}_{\mathbb{B}} = tt$,
$\mathsf{false}_{\mathbb{B}} = ff$,
$\neg_{\mathbb{B}} = \{tt \mapsto ff, ff \mapsto tt\}$

and

$\vee_{\mathbb{B}}$	tt	ff
tt	tt	tt
ff	tt	ff

$\wedge_{\mathbb{B}}$	tt	ff
tt	tt	ff
ff	ff	ff

$\Rightarrow_{\mathbb{B}}$	tt	ff
tt	tt	ff
ff	tt	tt

This is (essentially) the same as $A3$ in Example 2.2.3. Note that $A1$ can be turned into a (trivial) Boolean algebra in a similar way, but this is not the case with $A2$.

Exercise. Given a Boolean algebra B, define a relation $\leq_B \subseteq |B| \times |B|$ by $a \leq_B b$ iff $a \vee_B b = b$. Show that $\leq_B$ is a partial order with true_B and false_B as its greatest and least elements respectively, and with $a \vee_B b$ yielding the least upper bound of a, b and $a \wedge_B b$ yielding their greatest lower bound. (In fact, $\langle |B|, \leq_B \rangle$ is a distributive lattice with top and bottom elements and complement $\neg_B$.) □

Exercise 2.2.5. Show that all Boolean algebras (the models of BA as introduced in Exercise 2.2.4) satisfy the *de Morgan laws*:

$\forall p, q{:}bool \bullet \neg(p \vee q) = \neg p \wedge \neg q$
$\forall p, q{:}bool \bullet \neg(p \wedge q) = \neg p \vee \neg q$ □

The following characterisation of the expressive power of flat equational specifications is one of the classical theorems of universal algebra.

Definition 2.2.6 (Equationally definable class). A class $\mathscr{A}$ of Σ-algebras is *equationally definable* if $\mathscr{A} = Mod[\langle \Sigma, \Phi \rangle]$ for some set Φ of Σ-equations. □

Definition 2.2.7 (Variety). A class $\mathscr{A}$ of Σ-algebras is *closed under subalgebras* if for any $A \in \mathscr{A}$ and subalgebra B of A, $B \in \mathscr{A}$. Similarly, $\mathscr{A}$ is *closed under homomorphic images* if for any $A \in \mathscr{A}$ and Σ-homomorphism $h: A \to B$, $h(A) \in \mathscr{A}$, and $\mathscr{A}$ is *closed under products* if for any family $\langle A_i \in \mathscr{A} \rangle_{i \in I}$, $\prod \langle A_i \rangle_{i \in I} \in \mathscr{A}$.

A non-empty class of Σ-algebras which is closed under subalgebras, homomorphic images, and products is called a *variety*. □

Proposition 2.2.8. *Any equationally definable class $\mathscr{A}$ of Σ-algebras is a variety.* □

Exercise 2.2.9. Prove Proposition 2.2.8: show that for any presentation $\langle \Sigma, \Phi \rangle$, $Mod[\langle \Sigma, \Phi \rangle]$ is closed under subalgebras, homomorphic images and products. For example, formalise the following argument to show closure under subalgebras: if $A \models_\Sigma \varphi$ and B is a subalgebra of A then $B \models_\Sigma \varphi$ since removing values from the carriers of an algebra does not affect the truth of universally quantified assertions about its behaviour. Closure under products and under homomorphic images are not much more difficult to prove. □

Theorem 2.2.10 (Birkhoff's Variety Theorem [Bir35]). *If Σ is a signature with a finite set of sort names then a class $\mathscr{A}$ of Σ-algebras is a variety iff $\mathscr{A}$ is equationally definable.* □

The "if" part of this theorem is (a special case of) Proposition 2.2.8. A complete proof of the "only if" part is beyond the scope of this book; the curious reader should consult e.g. [Wec92].

Example 2.2.11. Consider the signature

$$\begin{array}{lll} \Sigma = & \textbf{sorts} & s \\ & \textbf{ops} & 0 : s \\ & & \_\_ \times \_\_ : s \times s \to s \end{array}$$

and the class $\mathscr{A}$ of Σ-algebras satisfying the familiar cancellation law:

if $a \neq 0$ and $a \times b = a \times c$ then $b = c$

The Σ-algebra A such that $|A|_s$ is the set of natural numbers and $\times_A$ is ordinary multiplication is in $\mathscr{A}$. The Σ-algebra B such that $|B|_s = \{0,1,2,3\}$ and $\times_A$ is multiplication modulo 4 is not in $\mathscr{A}$. (**Exercise:** Why not?) Since B is a homomorphic image of A, this shows that $\mathscr{A}$ is not a variety and hence is not equationally definable. □

Exercise 2.2.12. Formulate a definition of what it means for a class of Σ-algebras to be closed under homomorphic coimages. Are varieties closed under homomorphic coimages? □

Exercise 2.2.13. Formulate definitions of what it means for a class of Σ-algebras to be closed under quotients, and under isomorphisms. Show that closure under both quotients and isomorphisms is equivalent to closure under homomorphic images. □

The assumption in Theorem 2.2.10 that the set of sort names in Σ is finite cannot easily be omitted:

Exercise 2.2.14. A family $\mathscr{B}$ of Σ-algebras is *directed* if any two algebras $B_1, B_2 \in \mathscr{B}$ are subalgebras of some $B \in \mathscr{B}$. Define the *union* $\bigcup\mathscr{B}$ of such a family to be the least Σ-algebra such that each $B \in \mathscr{B}$ is a subalgebra of $\bigcup\mathscr{B}$ (the carrier of $\bigcup\mathscr{B}$ is the union of the carriers of all algebras in $\mathscr{B}$, and the values of operations on arguments are inherited from the algebras in $\mathscr{B}$; this is well-defined since $\mathscr{B}$ is directed). Prove that since we consider equations with finite sets of variables only, then for any presentation $\langle\Sigma,\Phi\rangle$, $Mod[\langle\Sigma,\Phi\rangle]$ is *closed under directed unions*, that is, given any *directed* family of algebras $\mathscr{B} \subseteq Mod[\langle\Sigma,\Phi\rangle]$, its union $\bigcup\mathscr{B}$ is also in $Mod[\langle\Sigma,\Phi\rangle]$.

A generalisation of Theorem 2.2.10 that we hint at here without a proof is that for *any* signature Σ, a class of Σ-algebras is equationally definable iff it is a variety that is closed under directed unions. □

Exercise 2.2.15. Consider a signature with an infinite set of sort names and no operations. Let $\mathscr{A}_{fin}$ be the class of all algebras over this signature that have non-empty carriers for a finite set of sorts only, and let $\mathscr{A}$ be the closure of $\mathscr{A}_{fin}$ under products and subalgebras (this adds algebras where the carrier of each sort is either a singleton or empty). Check that $\mathscr{A}$ is a variety. Prove, however, that $\mathscr{A}$ is not definable by any set of equations. HINT: Use Exercise 2.2.14. □

Exercise 2.2.16. Modify the definition of equation (Definition 2.1.1) so that infinite sets of variables are allowed; it is enough to consider sets of variables that are finite for each sort, but may be non-empty for infinitely many sorts. Extend the notion of satisfaction (Definition 2.1.2) to such generalised equations in the obvious way. Check that the class $\mathscr{A}$ defined in Exercise 2.2.15 is definable by such equations. HINT: Consider all equations of the form $\forall X \cup \{x,y{:}s\} \bullet x = y$, for all sorts s and sets X of variables such that $X_{s'} \neq \varnothing$ for infinitely many sorts s'.

Another generalisation of Theorem 2.2.10 that we want to hint at here is that for *any* signature Σ a class of Σ-algebras is definable by such generalised equations iff it is a variety. The proof of the "if" part is as easy as for ordinary equations (Proposition 2.2.8). The proof of the "only if" part is also quite similar as in the finitary case. □

A final remark to clarify the nuances in the many-sorted versions of Theorem 2.2.10 is that the theorem holds for *any* signature (also with an infinite set

of sort names) when we restrict attention to algebras with non-empty carriers of all sorts: all varieties of such algebras (with closure under subalgebras limited to subalgebras with non-empty carriers) are definable by equations with a finite set of variables.

2.3 Theories

Any given equationally definable class of algebras has many different presentations; in practice the choice of presentation is determined by various factors including the need for simplicity and understandability and the desire for elegance. On the other hand, such a class determines a single set of equations which uniquely identifies it, called its theory. Since this is an infinite set, it is not a useful way of presenting the class. However, it is a useful set to consider since it contains all axioms in all presentations of the class, together with all their consequences.

Throughout this section, let Σ be a signature.

Definition 2.3.1 ($Mod_\Sigma(\Phi)$, $Th_\Sigma(\mathscr{A})$, $Cl_\Sigma(\Phi)$ and $Cl_\Sigma(\mathscr{A})$). For any set Φ of Σ-equations, $Mod_\Sigma(\Phi)$ (the *models of* Φ) denotes the class of all Σ-algebras satisfying all the Σ-equations in Φ:

$$Mod_\Sigma(\Phi) = \{A \mid A \text{ is a } \Sigma\text{-algebra and } A \models_\Sigma \Phi\} \quad (= Mod[\langle \Sigma, \Phi \rangle]).$$

For any class $\mathscr{A}$ of Σ-algebras, $Th_\Sigma(\mathscr{A})$ (the *theory of* $\mathscr{A}$) denotes the set of all Σ-equations satisfied by each Σ-algebra in $\mathscr{A}$:

$$Th_\Sigma(\mathscr{A}) = \{\varphi \mid \varphi \text{ is a } \Sigma\text{-equation and } \mathscr{A} \models_\Sigma \varphi\}.$$

A set Φ of Σ-equations is *closed* if $\Phi = Th_\Sigma(Mod_\Sigma(\Phi))$. The *closure* of a set Φ of Σ-equations is the (closed) set $Cl_\Sigma(\Phi) = Th_\Sigma(Mod_\Sigma(\Phi))$. Analogously, a class $\mathscr{A}$ of Σ-algebras is *closed* if $\mathscr{A} = Mod_\Sigma(Th_\Sigma(\mathscr{A}))$, and the *closure* of $\mathscr{A}$ is $Cl_\Sigma(\mathscr{A}) = Mod_\Sigma(Th_\Sigma(\mathscr{A}))$. □

Proposition 2.3.2. *For any sets Φ and Ψ of Σ-equations and classes $\mathscr{A}, \mathscr{B}$ of Σ-algebras:*

1. *If $\Phi \subseteq \Psi$ then $Mod_\Sigma(\Phi) \supseteq Mod_\Sigma(\Psi)$.*
2. *If $\mathscr{B} \supseteq \mathscr{A}$ then $Th_\Sigma(\mathscr{B}) \subseteq Th_\Sigma(\mathscr{A})$.*
3. *$\Phi \subseteq Th_\Sigma(Mod_\Sigma(\Phi))$ and $Mod_\Sigma(Th_\Sigma(\mathscr{A})) \supseteq \mathscr{A}$.*
4. *$Mod_\Sigma(\Phi) = Mod_\Sigma(Th_\Sigma(Mod_\Sigma(\Phi)))$ and $Th_\Sigma(\mathscr{A}) = Th_\Sigma(Mod_\Sigma(Th_\Sigma(\mathscr{A}))))$.*
5. *$Cl_\Sigma(\Phi)$ and $Cl_\Sigma(\mathscr{A})$ are closed.*

Proof. **Exercise**. (HINT: Properties 4 and 5 follow from properties 1–3.) □

For any signature Σ, the functions Th_Σ and Mod_Σ constitute what is known in lattice theory as a Galois connection.

Definition 2.3.3 (Galois connection). A *Galois connection* is given by two partially ordered sets A and M (in Proposition 2.3.2, A is the set of all sets of Σ-equations, and M is the "set" of all classes of Σ-algebras, both ordered by inclusion) and maps $\_\_^*: A \to M$ and $\_\_^+: M \to A$ (here Mod_Σ and Th_Σ) satisfying properties corresponding to 2.3.2(1)–2.3.2(3). An element $a \in A$ (resp. $m \in M$) is called *closed* if $a = (a^*)^+$ (resp. $m = (m^+)^*$). □

Some useful properties — including ones corresponding to 2.3.2(4) and 2.3.2(5) — hold for any Galois connection.

Exercise 2.3.4. For any Galois connection and any $a, b \in A$ and $m \in M$, show that the following properties hold:

1. $a \leq_A m^+$ iff $a^* \geq_M m$.
2. If a and b are closed then $a \leq_A b$ iff $a^* \geq_M b^*$. (Show that the "if" part fails if a or b is not closed.)

Here, $\leq_A$ and $\leq_M$ are the orders on A and M respectively. □

Exercise 2.3.5. For any Galois connection such that A and M have binary least upper bounds ($\sqcup_A$, $\sqcup_M$) and greatest lower bounds ($\sqcap_A$, $\sqcap_M$), and for any $a, b \in A$, show that the following properties hold:

1. $(a \sqcup_A b)^* = a^* \sqcap_M b^*$.
2. $(a \sqcap_A b)^* \geq_M a^* \sqcup_M b^*$.

(HINT: $\sqcup_A$ satisfies the following properties for any $a, b, c \in A$:

- $a \leq_A a \sqcup_A b$ and $b \leq_A a \sqcup_A b$.
- If $a \leq_A c$ and $b \leq_A c$ then $a \sqcup_A b \leq_A c$.

and analogously for $\sqcap_A$, $\sqcup_M$ and $\sqcap_M$.) State and prove analogues to 1 and 2 for any $m, n \in M$, and instantiate all these general properties for the Galois connection between sets of Σ-equations and classes of Σ-algebras. □

Definition 2.3.6 (Semantic consequence). A Σ-equation φ is a *semantic consequence* of a set Φ of Σ-equations, written $\Phi \models_\Sigma \varphi$, if $\varphi \in Cl_\Sigma(\Phi)$ (equivalently, if $Mod_\Sigma(\Phi) \models_\Sigma \varphi$). □

Notation. We will write $\Phi \models \varphi$ instead of $\Phi \models_\Sigma \varphi$ when the signature Σ is obvious. □

The use of the double turnstile ($\models$) here is the same as its use in logic: $\Phi \models \varphi$ if the equation φ is satisfied in every algebra which satisfies all the equations in Φ. Here, Φ is a set of *assumptions* and φ is a *conclusion* which *follows from* Φ. We refer to this as *semantic* (or *model-theoretic*) consequence to distinguish it from a similar relation defined by means of "syntactic" inference rules in the next section.

Example 2.3.7. Recall Example 2.2.3. The exercise there shows:

$$\Phi\text{BOOL} \models_{\Sigma\text{BOOL}} \forall p{:}bool \bullet \neg(p \wedge \neg false) = \neg p$$
$$\Phi\text{BOOL} \not\models_{\Sigma\text{BOOL}} \forall p{:}bool \bullet \neg\neg p = p$$

Then, referring to Example 2.2.4, Exercise 2.2.5 shows that the de Morgan laws are semantical consequences of the set of axioms $\Phi\mathrm{BA}$. □

Exercise 2.3.8. Prove that semantic consequence is preserved by translation along signature morphisms: for any signature morphism $\sigma: \Sigma \to \Sigma'$, set Φ of Σ-equations, and Σ-equation φ,

$$\text{if } \Phi \models_\Sigma \varphi \text{ then } \sigma(\Phi) \models_{\Sigma'} \sigma(\varphi).$$

Equivalently, $\sigma(Cl_\Sigma(\Phi)) \subseteq Cl_{\Sigma'}(\sigma(\Phi))$. Show that the reverse inclusion does not hold. □

Exercise 2.3.9. Let $\sigma: \Sigma \to \Sigma'$ be a signature morphism and let Φ' be a closed set of Σ'-equations. Show that $\sigma^{-1}(\Phi')$ is a closed set of Σ-equations. □

See Section 4.2 for some further results on semantic consequence and translation along signature morphisms, presented in a more general context.

Definition 2.3.10 (Theory). A *theory* is a presentation $\langle \Sigma, \Phi \rangle$ such that Φ is closed. A presentation $\langle \Sigma, \Phi \rangle$ (where Φ need not be closed) *presents* the theory $\langle \Sigma, Cl_\Sigma(\Phi) \rangle$. A theory $\langle \Sigma, \Phi \rangle$ is sometimes referred to as a *Σ-theory*. □

A theory morphism between two theories is a signature morphism between their signatures that maps the equations in the source theory to equations belonging to the target theory.

Definition 2.3.11 (Theory morphism). For any theories $\langle \Sigma, \Phi \rangle$ and $\langle \Sigma', \Phi' \rangle$, a *theory morphism* $\sigma: \langle \Sigma, \Phi \rangle \to \langle \Sigma', \Phi' \rangle$ is a signature morphism $\sigma: \Sigma \to \Sigma'$ such that $\sigma(\varphi) \in \Phi'$ for every $\varphi \in \Phi$; if moreover σ is a signature inclusion $\sigma: \Sigma \hookrightarrow \Sigma'$ then $\sigma: \langle \Sigma, \Phi \rangle \hookrightarrow \langle \Sigma', \Phi' \rangle$ is a *theory inclusion*. □

Exercise 2.3.12. Let $\sigma: \langle \Sigma, \Phi \rangle \to \langle \Sigma', \Phi' \rangle$ and $\sigma': \langle \Sigma', \Phi' \rangle \to \langle \Sigma'', \Phi'' \rangle$ be theory morphisms. Show that $\sigma;\sigma': \Sigma \to \Sigma''$ is a theory morphism $\sigma;\sigma': \langle \Sigma, \Phi \rangle \to \langle \Sigma'', \Phi'' \rangle$. □

Proposition 2.3.13. *Let $\sigma: \Sigma \to \Sigma'$ be a signature morphism, Φ be a set of Σ-equations and Φ' be a set of Σ'-equations. Then the following conditions are equivalent:*

1. *σ is a theory morphism $\sigma: \langle \Sigma, Cl_\Sigma(\Phi) \rangle \to \langle \Sigma', Cl_{\Sigma'}(\Phi') \rangle$.*
2. *$\sigma(\Phi) \subseteq Cl_{\Sigma'}(\Phi')$.*
3. *For every $A' \in Mod_{\Sigma'}(\Phi')$, $A'|_\sigma \in Mod_\Sigma(\Phi)$.*

Proof. **Exercise**. (HINT: Use the Satisfaction Lemma, Lemma 2.1.8.) □

The fact that 2.3.13(2) implies 2.3.13(1) gives a shortcut for checking if a signature morphism is a theory morphism: one need only check, for each axiom in some *presentation* of the source theory, that the translation of that axiom is in the target theory. The equivalence between 2.3.13(1) and 2.3.13(3) is similar in spirit to the Satisfaction Lemma, demonstrating a perfect correspondence between translation

of syntax (axioms) along a signature morphism and translation of semantics (models) in the opposite direction. This equivalence shows that there is a model-level alternative to the axiom-level phrasing of Definition 2.3.11; in fact, we will take this alternative in the case of structured specifications (Chapter 5) where there is no equivalent axiom-level characterisation (Exercise 5.5.4).

Example 2.3.14. Let Σ be the signature

$$\begin{array}{lll}\Sigma = \textbf{sorts} & s, b \\ \quad \textbf{ops} & \mathit{ttr}: b \\ & \mathit{ffa}: b \\ & \mathit{not}: b \to b \\ & \mathit{and}: b \times b \to b \\ & \_\_ \leq \_\_ : s \times s \to b\end{array}$$

and recall the presentation $\mathrm{BOOL} = \langle \Sigma\mathrm{BOOL}, \Phi\mathrm{BOOL}\rangle$ from Example 2.2.3. Define a signature morphism $\sigma: \Sigma \to \Sigma\mathrm{BOOL}$ by

$$\begin{array}{l}\sigma_{\mathit{sorts}} = \{s \mapsto \mathit{bool}, b \mapsto \mathit{bool}\}, \\ \sigma_{\varepsilon,b} = \{\mathit{ttr} \mapsto \mathit{true}, \mathit{ffa} \mapsto \mathit{false}\}, \\ \sigma_{b,b} = \{\mathit{not} \mapsto \neg\}, \\ \sigma_{bb,b} = \{\mathit{and} \mapsto \wedge\}, \\ \sigma_{ss,b} = \{\leq \mapsto \Rightarrow\}.\end{array}$$

Let Φ be the set of Σ-equations

$$\Phi = \{\, \forall x{:}s \bullet x \leq x = \mathit{ttr},\ \forall p{:}b \bullet \mathit{and}(p, \mathit{ttr}) = p \,\}.$$

Then $Cl_\Sigma(\Phi)$ includes Σ-equations that were not in Φ, such as $\forall p{:}b, x{:}s \bullet \mathit{and}(p, x \leq x) = p$. Similarly, by Example 2.3.7, $Cl_{\Sigma\mathrm{BOOL}}(\Phi\mathrm{BOOL})$ includes the $\Sigma\mathrm{BOOL}$-equation $\forall p{:}\mathit{bool} \bullet \neg(p \wedge \neg \mathit{false}) = \neg p$, but it does *not* include $\forall p{:}\mathit{bool} \bullet \neg\neg p = p$. The presentations $\langle \Sigma, Cl_\Sigma(\Phi)\rangle$ and $\langle \Sigma\mathrm{BOOL}, Cl_{\Sigma\mathrm{BOOL}}(\Phi\mathrm{BOOL})\rangle$ are theories — the latter is the theory presented by BOOL. The signature morphism $\sigma: \Sigma \to \Sigma\mathrm{BOOL}$ is a theory morphism $\sigma: \langle \Sigma, Cl_\Sigma(\Phi)\rangle \to \langle \Sigma\mathrm{BOOL}, Cl_{\Sigma\mathrm{BOOL}}(\Phi\mathrm{BOOL})\rangle$.

Recalling Example 2.2.4, the theory presented by BA is $\langle \Sigma\mathrm{BA}, Cl_{\Sigma\mathrm{BA}}(\Phi\mathrm{BA})\rangle$, the theory of Boolean algebras, with $Cl_{\Sigma\mathrm{BA}}(\Phi\mathrm{BA})$ including for instance the de Morgan laws (Exercise 2.2.5). The obvious signature morphism $\iota: \Sigma\mathrm{BOOL} \to \Sigma\mathrm{BA}$ is a theory morphism $\iota: \langle \Sigma\mathrm{BOOL}, Cl_{\Sigma\mathrm{BOOL}}(\Phi\mathrm{BOOL})\rangle \to \langle \Sigma\mathrm{BA}, Cl_{\Sigma\mathrm{BA}}(\Phi\mathrm{BA})\rangle$.

These two theory morphisms can be composed, yielding the theory morphism $\sigma;\iota: \langle \Sigma, Cl_\Sigma(\Phi)\rangle \to \langle \Sigma\mathrm{BA}, Cl_{\Sigma\mathrm{BA}}(\Phi\mathrm{BA})\rangle$. □

Exercise 2.3.15. Give presentations $\langle \Sigma, \Phi\rangle$ and $\langle \Sigma', \Phi'\rangle$ and a theory morphism $\sigma: \langle \Sigma, Cl_\Sigma(\Phi)\rangle \to \langle \Sigma', Cl_{\Sigma'}(\Phi')\rangle$ such that $\sigma(\Phi) \not\subseteq \Phi'$. Note that this does *not* contradict the equivalence between 2.3.13(1) and 2.3.13(2). □

2.4 Equational calculus

As we have seen, each presentation $\langle \Sigma, \Phi \rangle$ determines a theory $\langle \Sigma, Cl_\Sigma(\Phi) \rangle$, where $Cl_\Sigma(\Phi)$ contains Φ together with all of its semantic consequences. An obvious question at this point is how to determine whether or not a given Σ-equation $\forall X \bullet t = t'$ belongs to the set $Cl_\Sigma(\Phi)$, i.e. how to decide if $\Phi \models_\Sigma \forall X \bullet t = t'$. The definition of $Cl_\Sigma(\Phi)$ does not provide an effective method: according to this, testing $\Phi \models_\Sigma \forall X \bullet t = t'$ involves constructing the (infinite!) class $Mod_\Sigma(\Phi)$ and checking whether or not $\forall X \bullet t = t'$ is satisfied by each of the algebras in this class, that is, checking for each algebra $A \in Mod_\Sigma(\Phi)$ and function $v : X \to |A|$ (there may be infinitely many such functions for a given A) that $t_A(v) = t'_A(v)$. An alternative is to proceed "syntactically" by means of *inference rules* which allow the elements of $Cl_\Sigma(\Phi)$ to be *derived* from the axioms in Φ via a sequence of formal proof steps.

Throughout this section, let Σ be a signature.

Definition 2.4.1 (Equational calculus). A Σ-equation φ is a *syntactic* (or *proof-theoretic*) *consequence* of a set Φ of Σ-equations, written $\Phi \vdash_\Sigma \varphi$, if this can be derived by application of the following inference rules:

Axiom: $$\frac{}{\Phi \vdash_\Sigma \forall X \bullet t = t'} \quad \forall X \bullet t = t' \in \Phi$$

Reflexivity: $$\frac{}{\Phi \vdash_\Sigma \forall X \bullet t = t} \quad X_s \subseteq \mathscr{X} \text{ for all } s \in S \text{ and } t \in |T_\Sigma(X)|$$

Symmetry: $$\frac{\Phi \vdash_\Sigma \forall X \bullet t = t'}{\Phi \vdash_\Sigma \forall X \bullet t' = t}$$

Transitivity: $$\frac{\Phi \vdash_\Sigma \forall X \bullet t = t' \qquad \Phi \vdash_\Sigma \forall X \bullet t' = t''}{\Phi \vdash_\Sigma \forall X \bullet t = t''}$$

Congruence: $$\frac{\Phi \vdash_\Sigma \forall X \bullet t_1 = t'_1 \qquad \cdots \qquad \Phi \vdash_\Sigma \forall X \bullet t_n = t'_n}{\Phi \vdash_\Sigma \forall X \bullet f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n)} \quad \begin{array}{l} f : s_1 \times \cdots \times s_n \to s \text{ in } \Sigma \text{ and} \\ t_i, t'_i \in |T_\Sigma(X)|_{s_i} \text{ for all } i \leq n \end{array}$$

Instantiation: $$\frac{\Phi \vdash_\Sigma \forall X \bullet t = t'}{\Phi \vdash_\Sigma \forall Y \bullet t[\theta] = t'[\theta]} \quad \theta : X \to |T_\Sigma(Y)|$$ □

Exercise 2.4.2 (Admissibility of weakening and cut). Prove that if $\Phi \vdash_\Sigma \forall X \bullet t = t'$ and $\Phi \subseteq \Phi'$ then $\Phi' \vdash_\Sigma \forall X \bullet t = t'$. (HINT: Simple induction on the structure of the derivation of $\Phi \vdash_\Sigma \forall X \bullet t = t'$.) This shows that the following rule is admissible[2]:

Weakening: $$\frac{\Phi \vdash_\Sigma \forall X \bullet t = t'}{\Phi \cup \Phi' \vdash_\Sigma \forall X \bullet t = t'}$$

[2] A rule is *admissible* in a formal system of rules if its conclusion is derivable in the system provided that all its premises are derivable. This holds in particular if the rule is *derivable* in the system, that is, if it can be obtained by composition of the rules in the system.

Prove that if $\Psi \vdash_\Sigma \varphi$ and $\{\varphi\} \cup \Phi \vdash_\Sigma \psi$ then $\Psi \cup \Phi \vdash_\Sigma \psi$. (HINT: Use induction on the structure of the derivation of $\{\varphi\} \cup \Phi \vdash_\Sigma \psi$; for the case of the axiom rule, use the fact that weakening is admissible.) This shows that the following rule is admissible:

$$\text{Cut: } \frac{\Psi \vdash_\Sigma \varphi \qquad \{\varphi\} \cup \Phi \vdash_\Sigma \psi}{\Psi \cup \Phi \vdash_\Sigma \psi}$$

Check that your proof can be generalised to show that if $\Phi \vdash \psi$ and $\Psi_\varphi \vdash \varphi$ for each $\varphi \in \Phi$ then $\bigcup_{\varphi \in \Phi} \Psi_\varphi \vdash \psi$. □

Exercise 2.4.3 (Consequence is preserved by translation). Show that for any signature morphism $\sigma : \Sigma \to \Sigma'$, set Φ of Σ-equations, and Σ-equation φ, if $\Phi \vdash_\Sigma \varphi$ then $\sigma(\Phi) \vdash_{\Sigma'} \sigma(\varphi)$. □

Example 2.4.4. Recall the presentation $\mathrm{BOOL} = \langle \Sigma\mathrm{BOOL}, \Phi\mathrm{BOOL} \rangle$ from Example 2.2.3. The following is a derivation of $\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet \neg(p \wedge \neg \mathit{false}) = \neg p$:

$$\dfrac{\dfrac{}{\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet \neg(p \wedge \neg \mathit{false}) = \neg(p \wedge \mathit{true})} \qquad \dfrac{\dfrac{}{\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet p \wedge \mathit{true} = p}}{\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet \neg(p \wedge \mathit{true}) = \neg p}}{\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet \neg(p \wedge \neg \mathit{false}) = \neg p}$$

where P is the derivation

$$\dfrac{\dfrac{}{\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet p = p} \qquad \dfrac{\dfrac{}{\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \neg \mathit{false} = \mathit{true}}}{\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet \neg \mathit{false} = \mathit{true}}}{\dfrac{\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet p \wedge \neg \mathit{false} = p \wedge \mathit{true}}{\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet \neg(p \wedge \neg \mathit{false}) = \neg(p \wedge \mathit{true})}}$$

Exercise. Tag each step above with the inference rule being applied. □

Exercise 2.4.5. Give a derivation of $\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}\mathit{bool} \bullet p \Rightarrow p = \mathit{true}$.

A considerably more serious challenge is to give derivations for the de Morgan laws from the axioms of Boolean algebra (see Example 2.2.4 and Exercise 2.2.5).

□

On its own, the equational calculus is nothing more than a game with symbols; its importance lies in the correspondence between the two relations $\models_\Sigma$ and $\vdash_\Sigma$. As we shall see, there is an exact correspondence: $\vdash_\Sigma$ is both *sound* and *complete* for $\models_\Sigma$. Soundness ($\Phi \vdash_\Sigma \varphi \Rightarrow \Phi \models_\Sigma \varphi$) is a vital property for any formal system: it ensures that the inference rules cannot be used to derive an incorrect result.

Theorem 2.4.6 (Soundness of equational calculus). *Let Φ be a set of Σ-equations and let φ be a Σ-equation. If $\Phi \vdash_\Sigma \varphi$ then $\Phi \models_\Sigma \varphi$.* □

Exercise 2.4.7. Prove Theorem 2.4.6. Use induction on the depth of the derivation of $\Phi \vdash_\Sigma \varphi$, showing that each rule in the system preserves the indicated property. □

Example 2.4.8. By Theorem 2.4.6, the formal derivation in Example 2.4.4 justifies the claim in Example 2.3.7 that $\Phi \mathrm{BOOL} \models_{\Sigma \mathrm{BOOL}} \forall p{:}bool \bullet \neg(p \wedge \neg false) = \neg p$. On the other hand, since $\Phi \mathrm{BOOL} \not\models_{\Sigma \mathrm{BOOL}} \forall p{:}bool \bullet \neg\neg p = p$, there can be no proof in the equational calculus for $\Phi \mathrm{BOOL} \vdash_{\Sigma \mathrm{BOOL}} \forall p{:}bool \bullet \neg\neg p = p$. □

It is a somewhat counter-intuitive fact (see [GM85]) that simplifying the calculus by omitting explicit quantifiers in equations yields an unsound system. This is due to the fact that algebras may have empty carrier sets. Any equation that includes a quantified variable $x{:}s$ will be satisfied by any algebra having an empty carrier for s, even if x appears on neither side of the equation. The instantiation rule is the only one that can be used to change the set of quantified variables; it is designed to ensure that quantified variables are eliminated only when it is sound to do so.

Exercise 2.4.9. Formulate a version of the equational calculus without explicit quantifiers on equations and show that it is unsound. (HINT: Consider the signature Σ with sorts s, s' and operations $f{:}s \to s'$, $a{:}s'$, $b{:}s'$, and set $\Phi = \{f(x) = a, f(x) = b\}$ of Σ-equations.

Show that $\Phi \vdash_\Sigma a = b$ in your version of the calculus. Then give a Σ-algebra $A \in Mod_\Sigma(\Phi)$ such that $A \not\models_\Sigma a = b$.) Pinpoint where this proof of unsoundness breaks down for the version of the equational calculus given in Definition 2.4.1. □

Exercise 2.4.10. Show that the equational calculus without explicit quantifiers is sound when the definition of Σ-algebra is changed to require all carrier sets to be non-empty, or when either of the following constraints on Σ is imposed:

1. Σ has only one sort.
2. All sorts in Σ are *non-void*: for each sort name s in Σ, $|T_\Sigma|_s \neq \varnothing$. □

Exercise 2.4.11. Give an example of a signature Σ which satisfies neither 2.4.10(1) nor 2.4.10(2), for which the equational calculus without explicit quantifiers is sound. □

Completeness ($\Phi \models_\Sigma \varphi \Rightarrow \Phi \vdash_\Sigma \varphi$) is typically more difficult to achieve than soundness: it means that the rules in the system are powerful enough to derive all correct results. It is not as important as soundness, in the sense that a complete

but unsound system is useless while (as we shall see in the sequel) a sound but incomplete system is often the best that can be obtained. The equational calculus happens to be complete for $\models_\Sigma$:

Theorem 2.4.12 (Completeness of equational calculus). *Let Φ be a set of Σ-equations and let φ be a Σ-equation. If $\Phi \models_\Sigma \varphi$ then $\Phi \vdash_\Sigma \varphi$.*

Proof sketch. Suppose $\Phi \models_\Sigma \forall X \bullet t = t'$. Define $\equiv \subseteq |T_\Sigma(X)| \times |T_\Sigma(X)|$ by $u \equiv u' \iff \Phi \vdash_\Sigma \forall X \bullet u = u'$; $\equiv$ is a Σ-congruence on $T_\Sigma(X)$. $T_\Sigma(X)/{\equiv} \models_\Sigma \Phi$ so $T_\Sigma(X)/{\equiv} \models_\Sigma \forall X \bullet t = t'$, and thus $t \equiv t'$, i.e. $\Phi \vdash_\Sigma \forall X \bullet t = t'$. □

Exercise 2.4.13. Fill in the gaps in the proof of Theorem 2.4.12. □

There are several different but equivalent versions of the equational calculus. The following exercise considers various alternatives to the congruence and instantiation rules.

Exercise 2.4.14. Show that the version of the equational calculus in Definition 2.4.1 is equivalent to the system obtained when the congruence and instantiation rules are replaced by the following single rule:

$$\text{Substitutivity: } \frac{\Phi \vdash_\Sigma \forall X \bullet t = t' \qquad \text{for each } x \in X,\ \Phi \vdash_\Sigma \forall Y \bullet \theta(x) = \theta'(x)}{\Phi \vdash_\Sigma \forall Y \bullet t[\theta] = t'[\theta']} \quad \theta, \theta' : X \to |T_\Sigma(Y)|$$

Show that this is equivalent to the system having the following more restricted version of the substitutivity rule:

$$\text{Substitutivity}': \frac{\Phi \vdash_\Sigma \forall X \cup \{x{:}s\} \bullet t = t' \qquad \Phi \vdash_\Sigma \forall Y \bullet u = u'}{\Phi \vdash_\Sigma \forall X \cup Y \bullet t[x \mapsto u] = t'[x \mapsto u']} \quad u, u' \in |T_\Sigma(Y)|_s$$

(HINT: The equivalence relies on the fact that the set of quantified variables in an equation is finite.) Finally, show that both of the following rules may be derived in any of these systems:

$$\text{Abstraction: } \frac{\Phi \vdash_\Sigma \forall X \bullet t = t'}{\Phi \vdash_\Sigma \forall X \cup Y \bullet t = t'} \quad Y_s \subseteq \mathscr{X} \text{ for all } s \in S$$

$$\text{Concretion: } \frac{\Phi \vdash_\Sigma \forall X \cup \{x{:}s\} \bullet t = t'}{\Phi \vdash_\Sigma \forall X \bullet t = t'} \quad t, t' \in |T_\Sigma(X)| \text{ and } |T_\Sigma(X)|_s \neq \varnothing$$

□

A consequence of the soundness and completeness theorems is that the equational calculus constitutes a *semi-decision procedure* for $\models_\Sigma$: enumerating all derivations will eventually produce a derivation for $\Phi \vdash_\Sigma \varphi$ if $\Phi \models_\Sigma \varphi$ holds, but if $\Phi \not\models_\Sigma \varphi$ then this procedure will never terminate. This turns out to be the best we can achieve:

Theorem 2.4.15. *There is no decision procedure for $\models_\Sigma$.*

Proof. Follows immediately from the undecidability of the word problem for semigroups [Pos47]. □

Mechanised proof search techniques can be applied with considerable success to the discovery of derivations (and under certain conditions, discussed in Section 2.6, a decision procedure *is* possible) but Theorem 2.4.15 shows that such techniques can provide no more than a partial solution.

2.5 Initial models

The class of algebras given by the loose semantics of a Σ-presentation contains too many algebras to be very useful in practice. In particular, Birkhoff's Variety Theorem guarantees that this class will always include degenerate Σ-algebras having a single value of each sort in Σ, as well as (nearly always) Σ-algebras that are not reachable. This unsatisfactory state of affairs is a consequence of the limited power of equational axioms. A standard way out is to take the so-called *initial semantics* of presentations, which selects a certain class of "best" models from among all those satisfying the axioms. Various alternatives to this approach will be presented in the sequel.

Throughout this section, let $\langle \Sigma, \Phi \rangle$ be a presentation.

Exercise 2.5.1. Verify the above claim concerning Birkhoff's Variety Theorem, being specific about the meaning of "nearly always". □

There are two features that render certain models of presentations unfit for use in practice. The mnemonic terms "junk" and "confusion" were coined in [BG81] to characterise these:

Definition 2.5.2 (Junk and confusion). Let A be a model of $\langle \Sigma, \Phi \rangle$. We say that A *contains junk* if it is not reachable, and that A *contains confusion* if it satisfies a ground Σ-equation that is not in $Cl_\Sigma(\Phi)$. □

The intuition behind these terms should be readily apparent: "junk" refers to useless values which could be discarded without being missed, and "confusion" refers to the values of two ground terms being unnecessarily identified (confused).

Example 2.5.3. Recall the presentation $\mathrm{BOOL} = \langle \Sigma\mathrm{BOOL}, \Phi\mathrm{BOOL} \rangle$ and its models $A1$, $A2$ and $A3$ from Example 2.2.3. $A1$ contains confusion ($A1 \models_{\Sigma\mathrm{BOOL}} \mathit{true} = \mathit{false} \notin Cl_{\Sigma\mathrm{BOOL}}(\Phi\mathrm{BOOL})$) but not junk; $A2$ contains junk (there is no ground $\Sigma\mathrm{BOOL}$-term t such that $t_{A2} = \spadesuit \in |A2|_{bool}$) but not confusion; $A3$ contains neither junk nor confusion. There are models of BOOL containing both junk and confusion. (**Exercise:** Find one.) □

Exercise 2.5.4. Consider the following specification of the natural numbers with addition:

$$\begin{array}{l}\textbf{spec}\ \mathrm{NAT} = \textbf{sorts}\ nat \\ \qquad\qquad \textbf{ops}\ \ 0 : nat \\ \qquad\qquad\qquad\ \ succ : nat \to nat \\ \qquad\qquad\qquad\ \ \_\_ + \_\_ : nat \times nat \to nat \\ \qquad\qquad \forall m, n : nat \bullet\ 0 + n = n \\ \qquad\qquad\qquad\qquad \bullet\ succ(m) + n = succ(m + n)\end{array}$$

List some of the models of NAT. Which of these contain junk and/or confusion? (NOTE: For reference later in this section, $\Sigma\mathrm{NAT}$ refers to the signature of NAT and $\Phi\mathrm{NAT}$ refers to its axioms.) □

Exercise 2.5.5. According to Exercise 1.3.5, surjective homomorphisms reflect junk. Show that injective homomorphisms preserve junk and reflect confusion, and that all homomorphisms preserve confusion. It follows that isomorphisms preserve and reflect junk and confusion. □

Examples like the ones above suggest that often the algebras of interest are those which contain neither junk nor confusion. Recall Exercise 1.4.14, which characterised reachable Σ-algebras as those which are isomorphic to a quotient of T_Σ. Accordingly, the algebras we want are all isomorphic to quotients of T_Σ; by Exercise 2.5.5 it is enough to consider just these quotient algebras themselves. Of course, not all quotients $T_\Sigma/\equiv$ will be models of $\langle \Sigma, \Phi \rangle$: this will only be the case when $\equiv$ identifies enough terms that the equations in Φ are satisfied. But if $\equiv$ identifies "too many" terms, $T_\Sigma/\equiv$ will contain confusion. There is exactly one Σ-congruence that yields a model of $\langle \Sigma, \Phi \rangle$ containing no confusion:

Definition 2.5.6 (Congruence generated by a set of equations). The relation $\equiv_\Phi \subseteq |T_\Sigma| \times |T_\Sigma|$ is defined by $t \equiv_\Phi t' \iff \Phi \models_\Sigma \forall \varnothing \bullet t = t'$, for all $t, t' \in |T_\Sigma|$. $\equiv_\Phi$ is called the *Σ-congruence generated by Φ*. □

Exercise 2.5.7. Prove that $\equiv_\Phi$ is a Σ-congruence on T_Σ. □

Theorem 2.5.8 (Quotient construction). *$T_\Sigma/\equiv_\Phi$ is a model of $\langle \Sigma, \Phi \rangle$ containing no junk and no confusion.* □

Exercise 2.5.9. Prove Theorem 2.5.8. HINT: Note that $T_\Sigma/\equiv_\Phi$ contains no junk by Exercise 1.4.14. Then show that for any term $t \in T_\Sigma(X)$ and substitution $\theta : X \to T_\Sigma$, $t_{T_\Sigma/\equiv_\Phi}(\theta') = [t[\theta]]_{\equiv_\Phi}$, where $\theta'(x) = [\theta(x)]_{\equiv_\Phi}$ for $x \in X$. Use this to show that $T_\Sigma/\equiv_\Phi$ satisfies all the equations in Φ and contains no confusion. □

Example 2.5.10. Recall the presentation $\mathrm{BOOL} = \langle \Sigma\mathrm{BOOL}, \Phi\mathrm{BOOL} \rangle$ from Example 2.2.3. The model $T_{\Sigma\mathrm{BOOL}}/\equiv_{\Phi\mathrm{BOOL}}$ of BOOL is defined as follows:

$$|T_{\Sigma\mathrm{BOOL}}/{\equiv_{\Phi\mathrm{BOOL}}}|_{bool} = \{[true]_{\equiv_{\Phi\mathrm{BOOL}}}, [false]_{\equiv_{\Phi\mathrm{BOOL}}}\}$$
$$true_{T_{\Sigma\mathrm{BOOL}}/{\equiv_{\Phi\mathrm{BOOL}}}} = [true]_{\equiv_{\Phi\mathrm{BOOL}}}$$
$$false_{T_{\Sigma\mathrm{BOOL}}/{\equiv_{\Phi\mathrm{BOOL}}}} = [false]_{\equiv_{\Phi\mathrm{BOOL}}}$$

$$\neg_{T_{\Sigma\mathrm{BOOL}}/{\equiv_{\Phi\mathrm{BOOL}}}} = \{[true]_{\equiv_{\Phi\mathrm{BOOL}}} \mapsto [false]_{\equiv_{\Phi\mathrm{BOOL}}}, [false]_{\equiv_{\Phi\mathrm{BOOL}}} \mapsto [true]_{\equiv_{\Phi\mathrm{BOOL}}}\}$$

$\wedge_{T_{\Sigma\mathrm{BOOL}}/{\equiv_{\Phi\mathrm{BOOL}}}}$	$[true]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[false]_{\equiv_{\Phi\mathrm{BOOL}}}$
$[true]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[true]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[false]_{\equiv_{\Phi\mathrm{BOOL}}}$
$[false]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[false]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[false]_{\equiv_{\Phi\mathrm{BOOL}}}$

$\Rightarrow_{T_{\Sigma\mathrm{BOOL}}/{\equiv_{\Phi\mathrm{BOOL}}}}$	$[true]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[false]_{\equiv_{\Phi\mathrm{BOOL}}}$
$[true]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[true]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[false]_{\equiv_{\Phi\mathrm{BOOL}}}$
$[false]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[true]_{\equiv_{\Phi\mathrm{BOOL}}}$	$[true]_{\equiv_{\Phi\mathrm{BOOL}}}$

where

$$[true]_{\equiv_{\Phi\mathrm{BOOL}}} = \{true, \neg false, true \wedge true, \neg(false \wedge true), \neg(false \wedge \neg false), false \Rightarrow false, \ldots\},$$
$$[false]_{\equiv_{\Phi\mathrm{BOOL}}} = \{false, \neg true, true \wedge false, \neg(true \wedge true), \neg(true \wedge \neg false), true \Rightarrow false, \ldots\}.$$

The carrier set $|T_{\Sigma\mathrm{BOOL}}/{\equiv_{\Phi\mathrm{BOOL}}}|_{bool}$ has just two elements since the axioms in $\Phi\mathrm{BOOL}$ can be used to reduce each ground $\Sigma\mathrm{BOOL}$-term to *true* or *false*, and $true \not\equiv_{\Phi\mathrm{BOOL}} false$. Note that the "syntactic" nature of $T_{\Sigma\mathrm{BOOL}}$ is preserved in $T_{\Sigma\mathrm{BOOL}}/{\equiv_{\Phi\mathrm{BOOL}}}$, e.g. for each $x \in [true]_{\equiv_{\Phi\mathrm{BOOL}}}$, "$\neg x$" $\in [false]_{\equiv_{\Phi\mathrm{BOOL}}} = \neg_{T_{\Sigma\mathrm{BOOL}}/{\equiv_{\Phi\mathrm{BOOL}}}}([true]_{\equiv_{\Phi\mathrm{BOOL}}})$. □

Exercise 2.5.11. Recall the presentation $\mathrm{NAT} = \langle \Sigma\mathrm{NAT}, \Phi\mathrm{NAT}\rangle$ given in Exercise 2.5.4. Construct the model $T_{\Sigma\mathrm{NAT}}/{\equiv_{\Phi\mathrm{NAT}}}$ of NAT. □

Exercise 2.5.12. Show that $\equiv_\Phi$ is the only Σ-congruence making Theorem 2.5.8 hold. □

The special properties of $T_\Sigma/{\equiv_\Phi}$ described by Theorem 2.5.8 can be captured very succinctly by saying that $T_\Sigma/{\equiv_\Phi}$ is a so-called *initial model* of $\langle \Sigma, \Phi\rangle$.

Definition 2.5.13 (Initial model of a presentation). A Σ-algebra A is *initial in* a class $\mathscr{A}$ of Σ-algebras if $A \in \mathscr{A}$ and for every $B \in \mathscr{A}$ there is a unique Σ-homomorphism $h: A \to B$. An *initial model of* $\langle \Sigma, \Phi\rangle$ is a Σ-algebra that is initial in $Mod[\langle \Sigma, \Phi\rangle]$. $IMod[\langle \Sigma, \Phi\rangle]$ is the class of all initial models of $\langle \Sigma, \Phi\rangle$. □

In the next chapter we will see that this definition can be generalised to a much wider context than that of algebras and homomorphisms.

Theorem 2.5.14 (Initial model theorem). *$T_\Sigma/{\equiv_\Phi}$ is an initial model of $\langle \Sigma, \Phi\rangle$.*

Proof sketch. $T_\Sigma/{\equiv_\Phi}$ is a model of $\langle \Sigma, \Phi\rangle$ by Theorem 2.5.8. Given $B \in Mod[\langle \Sigma, \Phi\rangle]$, let $\varnothing^\sharp: T_\Sigma \to B$ be the unique homomorphism from the algebra of ground Σ-terms to B. Since $B \models_\Sigma \Phi$, we have $\equiv_\Phi \subseteq K(\varnothing^\sharp)$, and by Exercise 1.3.20 there is a homomorphism $h: T_\Sigma/{\equiv_\Phi} \to B$, which is unique by Exercise 1.3.6. (**Exercise:** Fill in the gaps in this proof.) □

Example 2.5.15. Recall the presentation $\mathrm{BOOL} = \langle \Sigma\mathrm{BOOL}, \Phi\mathrm{BOOL}\rangle$ and its models $A1$, $A2$ and $A3$ from Example 2.2.3, and its model $T_{\Sigma\mathrm{BOOL}}/\equiv_{\Phi\mathrm{BOOL}}$ from Example 2.5.10, which is an initial model by Theorem 2.5.14. ΣBOOL-homomorphisms from $T_{\Sigma\mathrm{BOOL}}/\equiv_{\Phi\mathrm{BOOL}}$ to $A1$, $A2$ and $A3$ are as follows:

$$\begin{array}{ll} h1\colon T_{\Sigma\mathrm{BOOL}}/\equiv_{\Phi\mathrm{BOOL}} \to A1 & h1_{bool} = \{[true]_{\equiv_{\Phi\mathrm{BOOL}}} \mapsto \star, [false]_{\equiv_{\Phi\mathrm{BOOL}}} \mapsto \star\}, \\ h2\colon T_{\Sigma\mathrm{BOOL}}/\equiv_{\Phi\mathrm{BOOL}} \to A2 & h2_{bool} = \{[true]_{\equiv_{\Phi\mathrm{BOOL}}} \mapsto \clubsuit, [false]_{\equiv_{\Phi\mathrm{BOOL}}} \mapsto \heartsuit\}, \\ h3\colon T_{\Sigma\mathrm{BOOL}}/\equiv_{\Phi\mathrm{BOOL}} \to A3 & h3_{bool} = \{[true]_{\equiv_{\Phi\mathrm{BOOL}}} \mapsto 1, [false]_{\equiv_{\Phi\mathrm{BOOL}}} \mapsto 0\}. \end{array}$$

(**Exercise:** Check uniqueness.)

$A1$ is not an initial model: for example, $\nexists h\colon A1 \to A2$ and $\nexists h\colon A1 \to A3$. In general, models containing confusion cannot be initial since homomorphisms preserve confusion (Exercise 2.5.5). Similarly, $A2$ is not an initial model: for example, $\nexists h\colon A2 \to A3$, since there is no value in $|A3|_{bool}$ to which h can map the "extra" value $\spadesuit \in |A2|_{bool}$. On the other hand, $A3$ is initial: for example, $\exists! g1\colon A3 \to A1$ (where $g1_{bool}(1) = g1_{bool}(0) = \star$), $\exists! g2\colon A3 \to A2$ (where $g2_{bool}(1) = \clubsuit$ and $g2_{bool}(0) = \heartsuit$), and $\exists! g\colon A3 \to T_{\Sigma\mathrm{BOOL}}/\equiv_{\Phi\mathrm{BOOL}}$ (where $g_{bool}(1) = [true]_{\equiv_{\Phi\mathrm{BOOL}}}$ and $g_{bool}(0) = [false]_{\equiv_{\Phi\mathrm{BOOL}}}$). □

Exercise 2.5.16. Recall the model you constructed in Exercise 2.5.11 of the specification NAT of natural numbers with addition. Show that there is a unique homomorphism from this model to each of the models you considered in Exercise 2.5.4. □

Exercise 2.5.17. Using Theorem 2.5.14, show that T_Σ is an initial model of $\langle \Sigma, \varnothing\rangle$. Contemplate how this relates to Fact 1.4.4 and Definition 1.4.5. □

Exercise 2.5.18. Note that initial models of $\langle \Sigma, \Phi\rangle$ may have empty carriers for some sorts. Show that this is necessary: give an example of a presentation $\langle \Sigma, \Phi\rangle$ such that no algebra is initial in the class of its models that have non-empty carriers of all sorts. Link this with Exercise 1.2.3. □

Taking a presentation $\langle \Sigma, \Phi\rangle$ to denote the class $IMod[\langle \Sigma, \Phi\rangle]$ of its initial models is called taking its *initial semantics*. We know from Theorem 2.5.14 that $IMod[\langle \Sigma, \Phi\rangle]$ is never empty. Although the motivation for wishing to exclude models containing junk and confusion was merely to weed out certain kinds of degenerate cases, the effect of this constraint is to restrict attention to an isomorphism class of models:

Exercise 2.5.19. Show that any two initial models of a presentation are isomorphic. Conclude that the initial models of a presentation are exactly those containing no junk and no confusion. □

For some purposes, restricting to an isomorphism class of models is clearly inappropriate. The following exercise demonstrates what can go wrong.

Exercise 2.5.20. Consider the addition of a subtraction operation $-\colon nat \times nat \to nat$ to the specification NAT in Exercise 2.5.4, with the axioms $\forall m\colon nat \bullet m - 0 = m$ and $\forall m, n\colon nat \bullet succ(m) - succ(n) = m - n$. These axioms do not fix the value of $m - n$

when $n > m$; assume that we are willing to accept any value in this case, perhaps because we are certain for some reason that it will never arise. Construct an initial model of this specification. Why is this model unsatisfactory? Can you think of a better model? What is the problem with restricting to an isomorphism class of models of this specification? □

The phenomenon illustrated here arises in cases where operations are not defined in a *sufficiently complete* way. Roughly speaking, a definition of an operation is sufficiently complete when the value produced by the operation is defined for all of the possible values of its arguments. See Definition 6.1.22 below for a proper definition of this term in a more general context.

One may argue that Exercise 2.5.20 is unconvincing, since the lack of sufficient completeness arises there because we do not really need $m - n$ to be defined as a natural number when $n > m$, and that this can be dealt with using one of the approaches to partial functions below (Sections 2.7.3, 2.7.4, or 2.7.5). However, the same phenomenon arises in other cases as well:

Exercise 2.5.21. Give a specification of natural numbers with a function that for each natural number n chooses an arbitrary number that is greater than n. HINT: You may first extend the specification NAT of Exercise 2.5.4 with a sort *bool* with operations and axioms as in BOOL in Example 2.2.3, and add a binary operation $\_\_ < \_\_ : nat \times nat \to bool$ with the following axioms:

$\forall n{:}nat \bullet 0 < succ(n) = true$
$\forall m{:}nat \bullet succ(m) < 0 = false$
$\forall m, n{:}nat \bullet succ(m) < succ(n) = m < n$

The required function $ch{:}\, nat \to nat$ may now be constrained by the obvious axiom $\forall n{:}nat \bullet n < ch(n) = true$.

Clearly, the definition of *ch* cannot be sufficiently complete. Construct the initial model of the resulting specification and check that it is not satisfactory. Referring to other algebraic approaches presented in Sections 2.7.3, 2.7.4, and 2.7.5 below, check that none of them offers a satisfactory solution either. □

The above exercise indicates one of the most compelling reasons for considering alternatives to initial semantics: requiring specifications to define all operations in a sufficiently complete way is much too restrictive in many practical cases. Such a requirement is also undesirable for methodological reasons, since it forces the specifier of a problem to make decisions which are more appropriately left to the implementor.

The comments above notwithstanding, there are certain common situations in which initial semantics is appropriate and useful. In particular, the implicit "no junk" constraint conveniently captures the "that's all there is" condition which is needed e.g. in inductive definitions of syntax.

Example 2.5.22. Consider the following specification of syntax for simple arithmetic expressions:

$$\begin{array}{l}\textbf{spec}\ \textsc{Expr} = \textbf{sorts}\ \mathit{expr}\\ \qquad\qquad \textbf{ops}\quad x,y,0\colon \mathit{expr}\\ \qquad\qquad\qquad\ \ \mathit{plus},\mathit{minus}\colon \mathit{expr}\times\mathit{expr}\to\mathit{expr}\\ \qquad\qquad \forall e,e'\colon \mathit{expr}\bullet \mathit{plus}(e,e') = \mathit{plus}(e',e)\end{array}$$

The axiom requires the *syntax* of addition to be commutative. In the initial semantics of EXPR, the "no junk" condition ensures that the only expressions (value of sort *expr*) are those built from 0, *x* and *y* using *plus* and *minus*. The "no confusion" condition ensures that no undesired identification of expressions occurs: for example, the syntax of addition is not associative and the syntax of subtraction is not commutative. □

Exercise 2.5.23. Write a specification of (finite) sets of natural numbers. The operations should include $\varnothing\colon \mathit{set}$, $\mathit{singleton}\colon \mathit{nat}\to\mathit{set}$ and $\cup\colon \mathit{set}\times\mathit{set}\to\mathit{set}$. □

The "no junk" condition is more powerful than it might appear to be at first glance. Imposing the constraint that every value be expressible as a ground term makes it possible to use induction on the structure of terms to prove properties of all the values in an algebra. This means that for reasoning about models of specifications containing no junk, such as initial models, it is sound to add an induction rule scheme to the equational calculus presented in the previous section. Since the form of the induction rule scheme varies according to the signature of the specification at hand, this is best illustrated by means of examples.

Example 2.5.24. Recall the presentation $\mathrm{NAT} = \langle \Sigma\mathrm{NAT}, \Phi\mathrm{NAT}\rangle$ of natural numbers with addition given in Exercise 2.5.4. To simplify notation, let *x* and *y* stand for variable names such that $x{:}\mathit{nat}$ and $y{:}\mathit{nat}$ are not in $\Sigma\mathrm{NAT}$ and $x{:}\mathit{nat}$ does not appear in the $\mathit{sorts}(\Sigma\mathrm{NAT})$-sorted set of variables X used below. The following induction rule scheme is sound for reachable models of NAT (and for reachable models of all other ΣNAT-presentations):

$$\frac{\Phi \vdash_{\Sigma\mathrm{NAT}} P(0) \qquad \Phi\cup\{P(x)\} \vdash_{\Sigma\mathrm{NAT}\cup\{x:\mathit{nat}\}} P(\mathit{succ}(x)) \qquad \Phi\cup\{P(x),P(y)\} \vdash_{\Sigma\mathrm{NAT}\cup\{x,y:\mathit{nat}\}} P(x+y)}{\Phi \vdash_{\Sigma\mathrm{NAT}} \forall x{:}\mathit{nat}\bullet P(x)}$$

Here, $P(x)$ stands for a $\Sigma\mathrm{NAT}\cup\{x{:}\mathit{nat}\}$-equation $\forall X\bullet t = t'$; think of this as a ΣNAT-equation with free variable $x{:}\mathit{nat}$. Then $P(0)$ stands for the ΣNAT-equation $\forall X\bullet t[x\mapsto 0] = t'[x\mapsto 0]$, $P(\mathit{succ}(x))$ stands for the $\Sigma\mathrm{NAT}\cup\{x{:}\mathit{nat}\}$-equation $\forall X\bullet t[x\mapsto \mathit{succ}(x)] = t'[x\mapsto \mathit{succ}(x)]$ and analogously for $P(y)$ and $P(x+y)$, and $\forall x{:}\mathit{nat}\bullet P(x)$ stands for the ΣNAT-equation $\forall X\cup\{x{:}\mathit{nat}\}\bullet t = t'$. The following additional inference rule is needed to infer equations over $\Sigma\mathrm{NAT}\cup\{x{:}\mathit{nat}\}$ and $\Sigma\mathrm{NAT}\cup\{x,y{:}\mathit{nat}\}$ from ΣNAT-equations:

$$\frac{\Phi \vdash_{\Sigma} \forall X\bullet t = t'}{\Phi \vdash_{\Sigma\cup\Sigma'} \forall X\bullet t = t'}$$

Exercise. Show that adding the two inference rules above to the equational calculus gives a system that is sound for reachable models of ΣNAT-presentations.

The inference rule scheme above can be used for proving theorems such as associativity and commutativity of $+$. But note that the axioms for $+$ fully define it in terms of 0 and *succ*: it is possible to prove by induction on the structure of terms that for every ground $\Sigma\mathrm{NAT}$-term t there is a ground $\Sigma\mathrm{NAT}$-term t' such that t' does not contain the $+$ operation and $\Phi \vdash_{\Sigma\mathrm{NAT}} t = t'$. (**Exercise:** Prove it. Note that this is a proof at the meta-level *about* $\vdash$, not a derivation at the object level *using* $\vdash$.) This shows that the third premise of the above induction rule scheme is redundant. Eliminating it gives the following scheme, which is more obviously related to the usual form of induction for natural numbers:

$$\frac{\Phi \vdash_{\Sigma\mathrm{NAT}} P(0) \qquad \Phi \cup \{P(x)\} \vdash_{\Sigma\mathrm{NAT}\cup\{x:nat\}} P(succ(x))}{\Phi \vdash_{\Sigma\mathrm{NAT}} \forall x{:}nat \bullet P(x)}$$

Taking $P(x)$ to be $\forall n, p{:}nat \bullet x+(n+p) = (x+n)+p$, we have the following derivation, which proves that addition is associative in initial models of NAT (**Exercise:** Supply the derivations P_1 and P_2):

$$\frac{\begin{array}{cc} \Phi \vdash_{\Sigma\mathrm{NAT}} \forall n, p{:}nat \bullet 0+(n+p) = (0+n)+p & \Phi \cup \{\forall n, p{:}nat \bullet x+(n+p) = (x+n)+p\} \vdash_{\Sigma\mathrm{NAT}\cup\{x:nat\}} \forall n, p{:}nat \bullet succ(x)+(n+p) = (succ(x)+n)+p \\ P_1 & P_2 \end{array}}{\Phi \vdash_{\Sigma\mathrm{NAT}} \forall x, n, p{:}nat \bullet x+(n+p) = (x+n)+p}$$

Note that there are models of NAT containing junk which do not satisfy $\forall x, n, p{:}nat \bullet x+(n+p) = (x+n)+p$. Hence, this equation is not in $Cl_{\Sigma\mathrm{NAT}}(\Phi\mathrm{NAT})$ and induction is required for its derivation. □

Exercise 2.5.25. Recall the presentation $\mathrm{BOOL} = \langle \Sigma\mathrm{BOOL}, \Phi\mathrm{BOOL}\rangle$ from Example 2.2.3. Give an induction rule scheme that is sound for reachable models of $\Sigma\mathrm{BOOL}$-presentations. (HINT: There will be five premises, one for each operation in BOOL.) Show that three of the premises are redundant (HINT: eliminate one operation at a time), which gives the following rule scheme:

$$\frac{\Phi \vdash_{\Sigma\mathrm{BOOL}} P(true) \qquad \Phi \vdash_{\Sigma\mathrm{BOOL}} P(false)}{\Phi \vdash_{\Sigma\mathrm{BOOL}} \forall x{:}bool \bullet P(x)}$$

Use this to prove that $\forall p{:}bool \bullet \neg\neg p = p$ holds in initial models of BOOL. Prove that the axiom $\forall p{:}bool \bullet p \wedge \neg p = false$ is redundant for the initial semantics of BOOL,

that is:

$$\Phi\mathrm{BOOL} \setminus \{\forall p{:}bool \bullet p \wedge \neg p = false\} \vdash_{\Sigma\mathrm{BOOL}} \forall p{:}bool \bullet p \wedge \neg p = false. \qquad \square$$

Adding an induction rule scheme appropriate to the signature at hand to the equational calculus gives a system that is sound for reasoning about initial models of specifications, and is more powerful than the equational calculus on its own. However, the resulting system is not always complete. In fact, it turns out that completeness is unachievable in general: there is *no* sound proof system that is complete for reasoning about initial models of arbitrary specifications. In order to prove that this is the case, it is necessary to formalize what we mean by the term "proof system". For our purposes it will suffice to assume that any proof system has a recursively enumerable set of theorems. See [Chu56] for a discussion of the philosophical considerations (e.g. finiteness of proofs, decidability of the correctness of individual proof steps) underlying this assumption.

Theorem 2.5.26 (Incompleteness for initial semantics). *There is a presentation $\langle \Sigma, \Phi \rangle$ such that there is no proof system which is sound and complete with respect to satisfaction of equations in the class of initial models of $\langle \Sigma, \Phi \rangle$.*

Proof ([MS85]). As a consequence of Matiyasevich's theorem, the set of equations which hold in the standard model of the natural numbers (with 0, *succ*, $+$, $\times$ and $-$, such that $m - n = 0$ when $n \geq m$) is not recursively enumerable [DMR76, Sect. 8]. Therefore, this cannot be the set of theorems produced by any proof system. It is easy to construct a (single-sorted) presentation having this as an initial model. (**Exercise:** Construct it.) Since all the initial models of a presentation are isomorphic (Exercise 2.5.19) and since isomorphisms preserve and reflect satisfaction of equations (Exercise 2.1.5), this completes the proof. $\square$

The fact that completeness cannot be achieved is of no real importance in practice: the equational calculus together with induction is perfectly adequate for normal use. But the failure of completeness does mean that care must be taken to distinguish between satisfaction ($\models$) and provability ($\vdash$) in theoretical work. It is important to recognize that model-theoretic satisfaction is the relation of primary importance, since it embodies *truth*. Provability is merely an approximation to truth, albeit one that is of great importance for practical use since it is based on mechanical syntactic manipulation. The failure of completeness means that the approximation cannot be exact, but by being sound it errs on the side of safety.

Exercise 2.5.27. Show that the equational calculus (without added induction rule schemes) is complete with respect to satisfaction of *ground* equations in initial models of specifications. $\square$

The additional specification techniques introduced in Chapter 5 will lead to a widening of the gap between satisfaction and provability. In particular, even completeness with respect to satisfaction of ground equations will be impossible to retain.

A generalisation of the concept of initial model is needed to give a fully satisfactory specification of classes of models that are naturally parametric with respect

to some basic data. An example is the definition of terms in Section 1.4, which is parametric in an S-sorted set of variables. Another is the specification of sets (see Exercise 2.5.23): it should be possible to specify sets without building in a specification of the kind of values in the sets (in this case, natural numbers).

Exercise 2.5.28. Suppose that all information about the natural numbers is removed from the specification of sets you gave in Exercise 2.5.23, by deleting operations on natural numbers like *succ* and changing the sort name *nat* to *elem*. Construct an initial model of the resulting specification. Why is this model unsatisfactory? □

The required concept is that of a *free* model extending a given algebra, which captures the idea of initiality *relative to* a fixed part of the model. See Section 3.5 for the details, Section 4.3 for the use of this concept in the context of specifications, and Chapter 6 for much more on the general topic of parameterisation.

2.6 Term rewriting

Although there is no decision procedure for $\models_\Sigma$ (Theorem 2.4.15), there is a class of specifications for which consequence can be decided. The idea is similar to the one behind the strategy used in mathematics for proving that an equation follows from a set of equational axioms: one applies the axioms in an attempt to reduce both sides of the equation to a common result, and if this is successful then the equation follows from the axioms. An essential ingredient of this strategy is the use of equations as directed *simplification* or *rewrite rules*.

Throughout this section, let $\Sigma = \langle S, \Omega \rangle$ be a signature, and let X be an S-sorted set of variables such that $X_s \subseteq \mathscr{X}$ for all $s \in S$.

Assumption. For simplicity of presentation, we assume throughout this section that either Σ has only one sort, or all sorts in Σ are non-void (see Exercise 2.4.10). Under this assumption, the version of the equational calculus without explicit quantifiers is sound, and all references to the calculus below are to this version. See Exercises 2.6.11 and 2.6.26 for hints on how to do away with this assumption. □

Definition 2.6.1 (Context). A *Σ-context for sort $s \in S$* is a term $C \in |T_\Sigma(X \uplus \Box{:}s)|$ containing one occurrence of the distinguished variable $\Box$. We write $C[\,]$ to suggest that C should be viewed as a term with a hole in it. Substitution of a term $t \in |T_\Sigma(X)|_s$ in $C[\,]$ gives the term $C[\Box{:}s \mapsto t] \in |T_\Sigma(X)|$, written $C[t]$. □

Definition 2.6.2 (Rewrite rule). A *Σ-rewrite rule r of sort $s \in S$* consists of two Σ-terms $t, t' \in |T_\Sigma(X)|_s$, written $t \to t'$. The *Σ-equation determined by r* is $Eq(r) =_{\mathrm{def}} t = t'$; by the assumption, we can dispense with explicit quantification of variables in equations. A Σ-rewrite rule $r = t \to t'$ of sort s determines a set of *reduction steps* $C[t[\theta]] \to_r C[t'[\theta]]$ for all Σ-contexts $C[\,]$ for sort s and substitutions $\theta{:}X \to |T_\Sigma(X)|$; this defines the relation $\to_r \subseteq |T_\Sigma(X)| \times |T_\Sigma(X)|$, the *one-step reduction relation generated by r*. The inverse of one-step reduction $\to_r$ is *one-step expansion*, written ${}_r\!\leftarrow$. □

A reduction step $u \to_r u'$ according to a rewrite rule $r = t \to t'$ is an application of an *instance* $t[\theta] \to t'[\theta]$ of r to replace the *subterm* $t[\theta]$ of u (corresponding to the "hole" in $C[\,]$) by $t'[\theta]$. The subterm $t[\theta]$ of u is called a *redex* (short for "reducible expression").

Definition 2.6.3 (Term rewriting system). A *Σ-term rewriting system* R is a set of Σ-rewrite rules. The *set of Σ-equations determined by* R is $Eq(R) = \{Eq(r) \mid r \in R\}$. The *one-step reduction relation generated by* R is the relation

$$\to_R \quad = \quad \bigcup_{r \in R} \to_r \qquad (\subseteq |T_\Sigma(X)| \times |T_\Sigma(X)|).$$

The inverse of one-step reduction $\to_R$ is *one-step expansion*, written ${}_R\!\leftarrow$. □

Given a set Φ of Σ-equations, a Σ-term rewriting system R will be of greatest relevance to Φ when $Cl_\Sigma(\Phi) = Cl_\Sigma(Eq(R))$. One way to obtain such an R is to use the equations themselves as rewrite rules by selecting an *orientation* for each equation $t = t'$: either $t \to t'$ or $t' \to t$. For reasons that will become clear below, the most useful orientation is the one in which the right-hand side of the rule is "simpler" than the left-hand side. It is not always obvious how to measure simplicity of terms — in fact, this is a major issue in the theory of term rewriting — and sometimes there is no satisfactory orientation, as in the case of an equation such as $n + m = m + n$.

In the rest of this section, let R be a Σ-term rewriting system.

Definition 2.6.4 (Reduction $\to_R^*$ and convertibility $\sim_R$). The *reduction relation* $\to_R^* \subseteq |T_\Sigma(X)| \times |T_\Sigma(X)|$ *generated by* R is the transitive reflexive closure of $\to_R$. In other words, $t \to_R^* t'$ if $t = t'$ or there exist terms $t_1, \ldots, t_n \in |T_\Sigma(X)|$, $n \geq 0$, such that $t \to_R t_1 \to_R \cdots \to_R t_n \to_R t'$; then we say that t *reduces to* t'. The inverse of reduction $\to_R^*$ is *expansion*, written ${}_R^*\!\leftarrow$. The *convertibility relation* $\sim_R \subseteq |T_\Sigma(X)| \times |T_\Sigma(X)|$ *generated by* R is the symmetric transitive reflexive closure of $\to_R$. In other words, $t \sim_R t'$ if $t = t'$ or there exist terms $t_1, \ldots, t_n \in |T_\Sigma(X)|$, $n \geq 0$, such that $t \to_R t_1$ or $t \,{}_R\!\leftarrow t_1$, and $t_1 \to_R t_2$ or $t_1 \,{}_R\!\leftarrow t_2$, and …, and $t_n \to_R t'$ or $t_n \,{}_R\!\leftarrow t'$; then we say that t *converts to* t'. □

Exercise 2.6.5. Check that $\sim_R$ is a Σ-congruence on $T_\Sigma(X)$. □

Example 2.6.6. Recall the presentation $\mathrm{BOOL} = \langle \Sigma\mathrm{BOOL}, \Phi\mathrm{BOOL} \rangle$ from Example 2.2.3. The following $\Sigma\mathrm{BOOL}$-term rewriting system RBOOL obviously satisfies $Cl_{\Sigma\mathrm{BOOL}}(\Phi\mathrm{BOOL}) = Cl_{\Sigma\mathrm{BOOL}}(Eq(\mathrm{RBOOL}))$:

$$\mathrm{RBOOL} = \{\neg \mathit{true} \to \mathit{false},\ \neg \mathit{false} \to \mathit{true},\ p \wedge \mathit{true} \to p,\ p \wedge \mathit{false} \to \mathit{false},\\ p \wedge \neg p \to \mathit{false},\ p \Rightarrow q \to \neg(p \wedge \neg q)\}.$$

(Observe that in the rule $p \Rightarrow q \to \neg(p \wedge \neg q)$, the right-hand side is not obviously simpler than the left-hand side.) We have (for example):

$$\begin{aligned}\neg(p \wedge (\underline{q \Rightarrow \neg \mathit{false}})) &\to_{\mathrm{RBOOL}} \neg(p \wedge \neg(q \wedge \underline{\neg\neg \mathit{false}})) \\ &\to_{\mathrm{RBOOL}} \neg(p \wedge \neg(q \wedge \underline{\neg \mathit{true}})) \\ &\to_{\mathrm{RBOOL}} \neg(p \wedge \neg(\underline{q \wedge \mathit{false}})) \\ &\to_{\mathrm{RBOOL}} \neg(p \wedge \underline{\neg \mathit{false}}) \\ &\to_{\mathrm{RBOOL}} \neg(\underline{p \wedge \mathit{true}}) \\ &\to_{\mathrm{RBOOL}} \neg p\end{aligned}$$

(at each step, the redex reduced by the step is underlined) so $\neg(p \wedge (q \Rightarrow \neg \mathit{false})) \to^*_{\mathrm{RBOOL}} \neg p$, and

$$\begin{aligned}\neg(p \wedge (q \Rightarrow \mathit{false})) &\;{}_{\mathrm{RBOOL}}\!\leftarrow\; \neg(p \wedge (q \Rightarrow \neg \mathit{true})) \\ &\to_{\mathrm{RBOOL}} \neg(p \wedge \neg(q \wedge \neg\neg \mathit{true})) \\ &\to_{\mathrm{RBOOL}} \neg(p \wedge \neg(q \wedge \neg \mathit{false})) \\ &\;{}_{\mathrm{RBOOL}}\!\leftarrow\; \neg(p \wedge \neg((q \wedge \mathit{true}) \wedge \neg \mathit{false})) \\ &\to_{\mathrm{RBOOL}} \neg(p \wedge \neg((q \wedge \mathit{true}) \wedge \mathit{true})) \\ &\to_{\mathrm{RBOOL}} \neg(p \wedge \neg(q \wedge \mathit{true}))\end{aligned}$$

so $\neg(p \wedge (q \Rightarrow \mathit{false})) \sim_{\mathrm{RBOOL}} \neg(p \wedge \neg(q \wedge \mathit{true}))$. □

Exercise 2.6.7. Recall the presentation $\mathrm{NAT} = \langle \Sigma\mathrm{NAT}, \Phi\mathrm{NAT} \rangle$ given in Exercise 2.5.4. Give a $\Sigma\mathrm{NAT}$-term rewriting system RNAT such that $Cl_{\Sigma\mathrm{NAT}}(\Phi\mathrm{NAT}) = Cl_{\Sigma\mathrm{NAT}}(Eq(\mathrm{RNAT}))$, and practice reducing and converting some $\Sigma\mathrm{NAT}$-terms using RNAT. □

The convertibility relation generated by R coincides with equality provable from $Eq(R)$. This fact is captured by the following two theorems.

Theorem 2.6.8 (Soundness of convertibility). *If $t \sim_R t'$ then $Eq(R) \vdash_\Sigma t = t'$.*

Proof sketch. Consider a reduction step $C[t[\theta]] \to_r C[t'[\theta]]$. This corresponds to a derivation involving: an application of the axiom rule, to derive $Eq(R) \vdash t = t'$; an application of instantiation, to derive $Eq(R) \vdash t[\theta] = t'[\theta]$; and repeated applications of reflexivity and congruence, to derive $Eq(R) \vdash C[t[\theta]] = C[t'[\theta]]$. The definition of $\sim_R$ as the symmetric transitive reflexive closure of $\to_R$ corresponds directly to applications of the symmetry, transitivity and reflexivity rules. (**Exercise:** Fill in the gaps in this proof.) □

Lemma 2.6.9. *Suppose $t, t' \in |T_\Sigma(X)|_s$ for $s \in S$. If $t \sim_R t'$ then:*

1. *$C[t] \sim_R C[t']$ for any Σ-context $C[\,]$ for sort s.*
2. *$t[\theta] \sim_R t'[\theta]$ for any substitution $\theta : X \to |T_\Sigma(X)|$.*

Proof. **Exercise:** Do it. □

Theorem 2.6.10 (Completeness of convertibility). *If $Eq(R) \vdash_\Sigma t = t'$ then $t \sim_R t'$.*

Proof sketch. By induction on the depth of the derivation of $Eq(R) \vdash_\Sigma t = t'$. The most interesting case is when the last step is an application of the congruence rule:

$$\frac{Eq(R) \vdash_\Sigma t_1 = t'_1 \qquad \cdots \qquad Eq(R) \vdash_\Sigma t_n = t'_n}{Eq(R) \vdash_\Sigma f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n)}$$

where $f{:}s_1 \times \cdots \times s_n \to s$. By the inductive assumption, $t_1 \sim_R t_1'$ and $\ldots$ and $t_n \sim_R t_n'$. Then, by repeated application of Lemma 2.6.9(1), we have $f(t_1, t_2 \ldots, t_n) \sim_R f(t_1', t_2 \ldots, t_n) \sim_R \cdots \sim_R f(t_1', t_2' \ldots, t_n')$ (using first the context $f(\Box{:}s_1, t_2 \ldots, t_n)$, then $f(t_1', \Box{:}s_2, \ldots, t_n)$, then $\ldots$, then $f(t_1', t_2', \ldots, \Box{:}s_n)$). When the last step of the derivation of $Eq(R) \vdash_\Sigma t = t'$ is an application of the instantiation rule, the result follows directly by Lemma 2.6.9(2). (**Exercise:** Complete the proof.) □

Exercise 2.6.11. Try to get rid of the need for the assumption on Σ made at the beginning of this section in all the definitions and results above. This will involve rewriting terms of the form $(X)t$ using rewrite rules of the form $\forall X \bullet t \to t'$, in both cases with explicit variable declarations. □

Given the exact correspondence between convertibility and provable equality, a decision procedure for $t \sim_R t'$ amounts to a decision procedure for $\Phi \vdash_\Sigma t = t'$, provided $Cl_\Sigma(\Phi) = Cl_\Sigma(Eq(R))$. The problem with testing $t \sim_R t'$ by simply applying the definition is that the "path" from t to t' may include both reduction steps and expansion steps, and may be of arbitrary length. But when R satisfies certain conditions, it is sufficient to test just a *single* path having the special form $t \to_R^* t'' \,{}_R^*\!\leftarrow t'$, which yields a simple and efficient decision procedure for convertibility.

Definition 2.6.12 (Normal form). A Σ-term $t \in T_\Sigma(X)$ is a *normal form (for R)* if there is no term t' such that $t \to_R t'$. □

Definition 2.6.13 (Termination). A Σ-term rewriting system R is *terminating* (or *strongly normalising*) if there is no infinite reduction sequence $t_1 \to_R t_2 \to_R \cdots$; that is, whenever $t_1 \to_R t_2 \to_R \cdots$, there is some (finite) $n \geq 1$ such that t_n is a normal form. □

The usual way to show that a term rewriting system R is terminating is to demonstrate that each rule in R reduces the complexity of terms according to some carefully-chosen measure.

Definition 2.6.14 (Confluence). A Σ-term rewriting system R is *confluent* (or *Church-Rosser*) if whenever $t \to_R^* t_1$ and $t \to_R^* t_2$, there is a term t_3 such that $t_1 \to_R^* t_3$ and $t_2 \to_R^* t_3$. □

Definition 2.6.15 (Completeness). A Σ-term rewriting system R is *complete* if it is both terminating and confluent. □

Completeness of a term rewriting system should not be confused with completeness of a proof system, as in for example Theorem 2.6.10 above.

Exercise 2.6.16. Suppose that R is a complete Σ-term rewriting system, and let $t \in |T_\Sigma(X)|$ be a Σ-term. Show that there is a unique normal form $NF_R(t) \in |T_\Sigma(X)|$ such that $t \to_R^* NF_R(t)$.

HINT: An *abstract reduction system* consists of a set A together with a binary relation $\to \subseteq A \times A$. A Σ-term rewriting system R is a particular example, where $A = |T_\Sigma(X)|$ and $\to$ is $\to_R$. Concepts such as normal form and confluence make sense in the context of any abstract reduction system, and the required property holds in this more abstract setting. □

Example 2.6.17. The term rewriting system RBOOL from Example 2.6.6 is both terminating and confluent, and is therefore complete. As the reduction sequence in Example 2.6.6 shows, $NF_{\mathrm{RBOOL}}(\neg(p \wedge (q \Rightarrow \neg \mathit{false}))) = \neg p$.

The term rewriting system $\mathrm{RBOOL}' = \mathrm{RBOOL} \cup \{p \wedge q \to q \wedge p\}$ is not terminating: $p \wedge q \to_{\mathrm{RBOOL}'} q \wedge p \to_{\mathrm{RBOOL}'} p \wedge q \to_{\mathrm{RBOOL}'} q \wedge p \to_{\mathrm{RBOOL}'} \cdots$.

The term rewriting system $\mathrm{RBOOL}'' = \mathrm{RBOOL} \cup \{(p \wedge q) \wedge r \to p \wedge (q \wedge r)\}$ is not confluent: $(p \wedge \neg p) \wedge q \to_{\mathrm{RBOOL}''} \mathit{false} \wedge q$ and $(p \wedge \neg p) \wedge q \to_{\mathrm{RBOOL}''} p \wedge (\neg p \wedge q)$, and both $\mathit{false} \wedge q$ and $p \wedge (\neg p \wedge q)$ are normal forms. □

Exercise 2.6.18. Is your term rewriting system RNAT from Exercise 2.6.7 complete? If not, find an alternative term rewriting system for NAT that is complete. □

Exercise 2.6.19. A Σ-term rewriting system R is *weakly confluent* if whenever $t \to_R t_1$ and $t \to_R t_2$, there is a term t_3 such that $t_1 \to_R^* t_3$ and $t_2 \to_R^* t_3$. Find a term rewriting system that is weakly confluent but not confluent. (HINT: Weak confluence plus termination implies confluence, so don't bother looking at terminating term rewriting systems.) Weak confluence is a much easier condition to check than confluence, so the usual way to prove that a term rewriting system is confluent is to show that it is weakly confluent and terminating. □

In view of the obvious analogy between reduction and computation, $NF_R(t)$ can be thought of as the *value* of t; since $NF_R(t)$ need not be a ground term, this is a more general notion of computation than the usual one.

Exercise 2.6.20. Convince yourself that $NF_R\colon |T_\Sigma(X)| \to |T_\Sigma(X)|$ is computable for any finite complete term rewriting system R — perhaps try to implement it in your favourite programming language. □

Theorem 2.6.21 (Decision procedure for convertibility). *If R is complete, then $t \sim_R t'$ iff $NF_R(t) = NF_R(t')$.* □

Exercise 2.6.22. Prove Theorem 2.6.21. (HINT: The proof does not depend on the definition of $\to_R$, but only on the assumption that R is complete.) □

Since $t \sim_R t'$ iff $Eq(R) \vdash_\Sigma t = t'$ (by soundness and completeness of convertibility) iff $Eq(R) \models_\Sigma t = t'$ (by soundness and completeness of the equational calculus), Theorem 2.6.21 constitutes a decision procedure for consequence:

Corollary 2.6.23 (Decision procedure for $Eq(R) \models_\Sigma t = t'$). *If R is complete, then $Eq(R) \models_\Sigma t = t'$ iff $NF_R(t) = NF_R(t')$.* □

Example 2.6.24. Since the term rewriting system RBOOL from Example 2.6.6 is complete (see Example 2.6.17), Corollary 2.6.23 can be used to prove that $Eq(\mathrm{RBOOL}) \models_{\Sigma\mathrm{BOOL}} \neg(p \wedge (q \Rightarrow \neg \mathit{false})) = p \Rightarrow (p \wedge \neg p)$: $NF_{\mathrm{RBOOL}}(\neg(p \wedge (q \Rightarrow \neg \mathit{false}))) = \neg p = NF_{\mathrm{RBOOL}}(p \Rightarrow (p \wedge \neg p))$. Since $Cl_{\Sigma\mathrm{BOOL}}(\Phi\mathrm{BOOL}) = Cl_{\Sigma\mathrm{BOOL}}(Eq(\mathrm{RBOOL}))$, this proves that $\Phi\mathrm{BOOL} \models_{\Sigma\mathrm{BOOL}} \neg(p \wedge (q \Rightarrow \neg \mathit{false})) = p \Rightarrow (p \wedge \neg p)$.

Exercise. Give a derivation of $\Phi\mathrm{BOOL} \vdash_{\Sigma\mathrm{BOOL}} \neg(p \wedge (q \Rightarrow \neg \mathit{false})) = p \Rightarrow (p \wedge \neg p)$ in the equational calculus. Compare this with the above proof. □

Exercise 2.6.25. Recall your complete term rewriting system for NAT from Exercise 2.6.18. Use this to prove that $\Phi\mathrm{NAT} \models_{\Sigma\mathrm{NAT}} succ(succ(0)) + succ(n) = succ(succ(succ(n)))$, and that $\Phi\mathrm{NAT} \not\models_{\Sigma\mathrm{NAT}} succ(m) + succ(n) = succ(succ(m + n))$, □

Exercise 2.6.26. Let $t \to t'$ be a Σ-rewrite rule of sort s. The following restrictions are often imposed:

- $t \notin X_s$; and
- $FV(t') \subseteq FV(t)$.

Show that, if these restrictions are imposed on rewrite rules, then Corollary 2.6.23 holds even without the assumption on Σ made at the beginning of this section. (These restrictions seem harmless since almost no complete term rewriting system contains rules that violate them.) □

Exercise 2.6.27. Equality of terms in the equational theory of a rewriting systems is also decidable under somewhat weaker requirements than those in Corollary 2.6.23. A term-rewriting system R is *weakly normalising* if for each term t there is a finite reduction sequence in R leading from t to a normal form. R is *semi-complete* if it is weakly normalising and confluent.

Generalising Exercise 2.6.16, show that if R is a semi-complete Σ-term rewriting system, then for any Σ-term $t \in |T_\Sigma(X)|$ there is a unique normal form $NF_R(t) \in |T_\Sigma(X)|$ such that $t \to_R^* NF_R(t)$. Moreover, convince yourself that the function $NF_R : |T_\Sigma(X)| \to |T_\Sigma(X)|$ is then computable. Finally, show that the property captured by Corollary 2.6.23 holds for all semi-complete term rewriting systems R. □

By Corollary 2.6.23, the problem of deciding consequence $\Phi \models_\Sigma \varphi$ is reduced to the problem of finding a finite complete term rewriting system R such that $Cl_\Sigma(\Phi) = Cl_\Sigma(Eq(R))$. Clearly, by Theorem 2.4.15, this is not always possible. But the *Knuth-Bendix completion algorithm* can sometimes be used to produce such an R given Φ together with an order relation on terms. The algorithm works by pinpointing causes of failure of (weak) confluence and adding rules to correct them, where the supplied term ordering is used to orient these new rules. The algorithm is iterative and may fail to terminate; it may also fail because the ordering supplied is inadequate.

The Knuth-Bendix completion algorithm can also be used to reason about initial models of specifications, using a method known as *inductionless induction* or *proof by consistency*. This method is based on the observation that an equation $t = t'$ holds in the initial models of $\langle \Sigma, \Phi \rangle$ iff there is no ground equation $s = s'$ such that $\Phi \not\models s = s'$ and $\Phi \cup \{t = t'\} \models s = s'$. (**Exercise:** Prove this fact.) Given a complete term rewriting system R such that $Cl_\Sigma(\Phi) = Cl_\Sigma(Eq(R))$ (perhaps produced using the Knuth-Bendix algorithm), the Knuth-Bendix algorithm is used to produce a complete term rewriting system R' for $\Phi \cup \{t = t'\}$ by extending R. It is then possible to test if R and R' have the same normal forms for ground Σ-terms; if so, then $t = t'$ holds in the initial models of $\langle \Sigma, \Phi \rangle$.

2.7 Fiddling with the definitions

In principle, the specification framework presented in the preceding sections is powerful enough for any conceivable computational application. This is made precise by a theorem in [BT87] (cf. [Vra88]) which states that for every reachable *semi-computable* Σ-algebra A there is a presentation $\langle \Sigma', \Phi' \rangle$ with finite Φ' such that $A = A'|_\Sigma$ for some initial model $A' \in IMod[\langle \Sigma', \Phi' \rangle]$. (See [BT87] for the definition of semi-computable algebra.) In spite of this fact, there are several reasons why this framework is inconvenient for use in practice.

One deficiency becomes apparent as soon as one attempts to write specifications that are somewhat larger than the examples we have seen so far. In order to be understandable and usable, large specifications must be built up incrementally from smaller specifications. Specification mechanisms designed to cope with such problems of scale are presented in Chapter 5. These methods also solve the problem illustrated by Exercise 2.5.20, see Exercise 5.1.11.

Another difficulty arises from the relatively low level of equational logic as a language for describing constraints to be satisfied by the operations of an algebra. When using equational axioms, it is often necessary to write a dozen equations to express a property that can be formulated much more clearly using a single axiom in some more powerful logic. Some properties that are easy to express in more powerful systems are not expressible at all using equations. Similar awkwardness is caused by the limitations of the type system used here, in comparison with the polymorphic type systems of modern programming languages such as Standard ML [Pau96]. Finally, the present framework is only able to cope conveniently with algebras comprised of *total* and *deterministic* functions operating on data values built by *finitary* compositions of such functions, a limitation which rules out its use for very many programs of interest.

All these difficulties can be addressed by making appropriate modifications to the standard framework presented in the preceding sections. An example was already given in Section 1.5.2 where it was shown how signature morphisms could be replaced by derived signature morphisms. This section is devoted to a sketch of some other possible modifications. The presentation is very brief and makes no attempt to be truly comprehensive; the interested reader will find further details (and further citations) in the cited references.

2.7.1 Conditional equations

The most obvious kind of modification to make is to replace the use of equational axioms by formulae in a more expressive language. Some care is required since a number of the results presented above depend on the use of equational axioms. A relatively unproblematic choice is to use equations that apply only when certain pre-conditions (expressed as equations) are satisfied.

Let $\Sigma = \langle S, \Omega \rangle$ be a signature.

Definition 2.7.1 (Conditional equation). A *(positive) conditional Σ-equation* $\forall X \bullet t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t_0 = t'_0$ consists of:

- a finite S-sorted set X (of variables), such that $X_s \subseteq \mathscr{X}$ for all $s \in S$; and
- for each $0 \leq j \leq n$ (where $n \geq 0$), two Σ-terms $t_j, t'_j \in |T_\Sigma(X)|_{s_j}$ for some sort $s_j \in S$.

A Σ-algebra A *satisfies* a conditional Σ-equation $\forall X \bullet t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t_0 = t'_0$ if for every (S-sorted) function $v{:}X \to |A|$, if $(t_1)_A(v) = (t'_1)_A(v)$ and $\ldots$ and $(t_n)_A(v) = (t'_n)_A(v)$ then $(t_0)_A(v) = (t'_0)_A(v)$. □

Note that variables in the conditions ($t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n$) that do not appear in the consequent ($t_0 = t'_0$) can be seen as existentially quantified: for example, the conditional equation $\forall a, b{:}t \bullet a \times b = 1 \Rightarrow a \times a^{-1} = 1$ is equivalent to the formula $\forall a{:}t \bullet (\exists b{:}t \bullet a \times b = 1) \Rightarrow a \times a^{-1} = 1$ in ordinary first-order logic.

Exercise 2.7.2. Define the translation of conditional Σ-equations by a signature morphism $\sigma{:}\Sigma \to \Sigma'$. □

The remaining definitions of Sections 2.1–2.5 require only superficial changes, and most results go through with appropriate modifications.

Let $\langle \Sigma, \Phi \rangle$ be a presentation, where Φ is a set of conditional Σ-equations. $Mod[\langle \Sigma, \Phi \rangle]$ is not always a variety, as is (almost) shown by Example 2.2.11; in this sense, the power of conditional equations is strictly greater than that of ordinary equations.

Exercise 2.7.3. The cancellation law given in Example 2.2.11 is not a conditional equation. Give a version of this example that uses only conditional equations. (HINT: Equality can be axiomatized as an operation $eq{:}s \times s \to bool$.) □

In spite of this increase in expressive power, there is a proof system that is sound and complete with respect to conditional equational consequence [Sel72], and the quotient construction can be used to construct an initial model of $\langle \Sigma, \Phi \rangle$ [MT92] (cf. Lemma 3.3.12 below). Term rewriting with conditional rewrite rules is possible, but there are some complications, see [Klo92] and [Mid93].

Exercise 2.7.4. [Sel72] gives a proof system that is sound and complete for conditional equational consequence in the single-sorted case. Extend this to the many-sorted case, where explicit quantifiers are required for the same reason as in the equational calculus. □

Exercise 2.7.5. Recall Exercise 2.5.21 concerning the specification of a function $ch{:}nat \to nat$ that for each natural number n chooses an arbitrary number that is greater than n. Modify this, using a conditional equation to make ch choose an arbitrary number that is *less* than n when $0 < n$. □

Example 2.7.6. Let $\mathrm{HA} = \langle \Sigma\mathrm{HA}, \Phi\mathrm{HA} \rangle$ be the following presentation.[3]

[3] We use the same symbol $\Rightarrow$ for implication in conditional equations and for an operation in the presentation below — the usual symbols are used for other propositional connectives as well, as in Example 2.2.4. We use extra space around implication in the conditional equations below in order to make them easier to read.

spec $\mathrm{HA} =$ **sorts** *bool*
ops $\mathsf{true}: bool$
$\mathsf{false}: bool$
$\neg\_: bool \to bool$
$\_\vee\_: bool \times bool \to bool$
$\_\wedge\_: bool \times bool \to bool$
$\_\Rightarrow\_: bool \times bool \to bool$
$\forall p, q, r: bool$
- $p \vee (q \vee r) = (p \vee q) \vee r$
- $p \wedge (q \wedge r) = (p \wedge q) \wedge r$
- $p \vee q = q \vee p$
- $p \wedge q = q \wedge p$
- $p \vee (p \wedge q) = p$
- $p \wedge (p \vee q) = p$
- $p \vee \mathsf{true} = \mathsf{true}$
- $p \vee \mathsf{false} = p$
- $(p \vee (r \wedge q) = p) \;\Rightarrow\; ((q \Rightarrow p) \vee r = (q \Rightarrow p))$
- $((q \Rightarrow p) \vee r = (q \Rightarrow p)) \;\Rightarrow\; (p \vee (r \wedge q) = p)$
- $\neg p = (p \Rightarrow \mathsf{false})$

Models of HA are called *Heyting algebras*.

Exercise. Recall the presentation BA of Boolean algebras in Example 2.2.4. Show that every Boolean algebra is a Heyting algebra. Then repeat the exercise in Example 2.2.4, building for every Heyting algebra H a lattice $\langle |H|, \leq_H \rangle$ with top and bottom elements. Check that the conditional axioms concerning the implication $\Rightarrow$ can now be captured by requiring that $r \wedge q \leq_H p$ is equivalent to $r \leq_H q \Rightarrow p$. Show that the lattice is distributive.

Give an example of a Heyting algebra that is not Boolean. Check which of the axioms of the presentation BA do not follow from HA.

Prove that an *equational* presentation with the same models as HA can be given. HINT: Use Theorem 2.2.10. Or consider the following properties of the implication: $p \Rightarrow p = \mathsf{true}$, $q \wedge (q \Rightarrow p) = q \wedge p$, $p \vee (q \Rightarrow p) = q \Rightarrow p$, and $q \Rightarrow (p \wedge r) = (q \Rightarrow p) \wedge (q \Rightarrow r)$. □

2.7.2 Reachable semantics

In Section 2.5, the motivation given for taking a presentation $\langle \Sigma, \Phi \rangle$ to denote the class $IMod[\langle \Sigma, \Phi \rangle]$ of its initial models was the desire to exclude models containing junk and confusion. The need to exclude models containing confusion stems mainly from the use of equational axioms, which make it impossible to rule out degenerate models having a single value of each sort in Σ. If a more expressive language is used for axioms, or if degenerate models are ruled out by some other means, then models containing confusion need not be excluded.

Example 2.7.7. Consider the following specification of sets of natural numbers (a variant of the one in Exercise 2.5.23):

spec SETNAT = **sorts** $bool, nat, set$
ops $true : bool$
$false : bool$
$\_\_ \vee \_\_ : bool \times bool \to bool$
$0 : nat$
$succ : nat \to nat$
$eq : nat \times nat \to bool$
$\varnothing : set$
$add : nat \times set \to set$
$\_\_ \in \_\_ : nat \times set \to bool$
$\forall p : bool, m, n : nat, S : set$
- $p \vee true = true$
- $p \vee false = p$
- $eq(n,n) = true$
- $eq(0, succ(n)) = false$
- $eq(succ(n), 0) = false$
- $eq(succ(m), succ(n)) = eq(m,n)$
- $n \in \varnothing = false$
- $m \in add(n,S) = eq(m,n) \vee m \in S$

There are many different models of SETNAT, including algebras having a single value of each sort. Suppose we restrict attention to algebras that do not satisfy the equation $\forall \varnothing \bullet true = false$; this excludes such degenerate models (see the exercise below). Consider the following two equations:

Commutativity of *add*: $\forall m, n : nat, S : set \bullet add(m, add(n,S)) = add(n, add(m,S))$
Idempotency of *add*: $\forall n : nat, S : set \bullet add(n, add(n,S)) = add(n,S)$

The models of SETNAT that do not satisfy $\forall \varnothing \bullet true = false$ may be classified according to which of these two equations they satisfy.

"List-like" algebras: *add* is neither commutative nor idempotent.
"Set-like" algebras: *add* is both commutative and idempotent.
"Multiset-like" algebras: *add* is commutative but not idempotent.
"List-like" algebras without repeated adjacent entries: *add* is idempotent but not commutative.

There are also "hybrid" models of SETNAT, e.g. those in which *add* is commutative but is only idempotent for $n \neq 0$. The initial models of SETNAT are "list-like" algebras. Adding the commutativity and idempotency requirements to SETNAT as additional axioms would eliminate all but the "set-like" algebras.

Exercise. Show that restricting attention to models of SETNAT that do not satisfy the equation $\forall \varnothing \bullet true = false$ eliminates all but "sensible" realisations of sets of natural numbers, by forcing $eq(succ^m(0), succ^n(0)) = true$ iff $m = n$ iff $succ^m(0) =$

$succ^n(0)$, and $a \in add(a_1, add(a_2, \ldots, add(a_p, \varnothing)\ldots)) = true$ iff $eq(a,a_1) = true$ or $\ldots$ or $eq(a,a_p) = true$, for $m,n,p \geq 0$. Note that m,n and p are ordinary integers here, *not* values of the sort *nat*, and $succ^m(0)$ means $\underbrace{succ(\ldots succ(}_{m \text{ times}}0)\ldots)$. □

Consideration of examples like the one above suggests various alternatives to taking the initial semantics of specifications. One choice is to require signatures to include the sort *bool* and the constants *true* and *false*, and to exclude models satisfying $\forall \varnothing \bullet true = false$. This might be termed taking the *standard loose semantics* of specifications. Another choice is to additionally exclude models containing junk:

Definition 2.7.8 (Reachable semantics). Let $\Sigma = \langle S, \Omega \rangle$ be a signature such that $bool \in S$ and $true{:}bool$ and $false{:}bool$ are in Ω. A *reachable standard model* of a presentation $\langle \Sigma, \Phi \rangle$ is a reachable Σ-algebra A such that $A \models_\Sigma \Phi$ and $A \not\models_\Sigma \forall \varnothing \bullet true = false$. $RMod[\langle \Sigma, \Phi \rangle]$ is the class of all reachable standard models of $\langle \Sigma, \Phi \rangle$. Taking $\langle \Sigma, \Phi \rangle$ to denote $RMod[\langle \Sigma, \Phi \rangle]$ is called taking its *reachable semantics*. □

The motivation for excluding models containing junk is the same as in the case of initial semantics. $RMod[\langle \Sigma, \Phi \rangle]$ is not always an isomorphism class of models, as Example 2.7.7 demonstrates (the classification given there was for *all* models that do not satisfy $\forall \varnothing \bullet true = false$, but the same applies to the reachable models in this class). There is still a problem when operations are not defined in a sufficiently complete way, although the problem is less severe than in the case of initial semantics.

Exercise 2.7.9. Reconsider the problem posed in Exercise 2.5.20, by writing a reachable model specification of natural numbers including a subtraction operation $\_\_ - \_\_ : nat \times nat \to nat$ with the axioms $\forall m{:}nat \bullet m - 0 = m$ and $\forall m,n{:}nat \bullet succ(m) - succ(n) = m - n$. Recall from Exercise 2.5.20 the assumption that we are willing to accept any value for $m - n$ when $n > m$, which is why the axioms do not constrain the value of $m - n$ in this case. List some of the reachable standard models of this specification, and decide whether the models you considered in Exercise 2.5.20 are reachable standard models (ignoring the difference in signatures). From an intuitive point of view, is this an adequate class of models for this specification? □

Exercise 2.7.10. Definition 2.7.8 permits algebras $A \in RMod[\langle \Sigma, \Phi \rangle]$ with values of sort *bool* other than $true_A$ and $false_A$. This is ruled out if all operations delivering results in sort *bool* are defined in a sufficiently complete way to yield either *true* or *false* on each argument that is definable by a ground term. Check that the specification SETNAT in Example 2.7.7 ensures this property and so all of its reachable models have a two-element carrier of sort *bool*. Give an example of a specification for which this is not the case. □

The equational calculus is sound for reasoning about the reachable standard models of presentations, since $RMod[\langle \Sigma, \Phi \rangle] \subseteq Mod[\langle \Sigma, \Phi \rangle]$ for any presentation $\langle \Sigma, \Phi \rangle$. It is sound to add induction rule schemes such as those given in Section 2.5; these are

sound for any class of reachable models. Completeness is unachievable, for exactly the same reason as in the case of initial semantics; the proof of Theorem 2.5.26 can be repeated in this context almost without change. Finally, the techniques of term rewriting presented in Section 2.6 remain sound.

Initial semantics cannot be used for specifications with axioms that are more expressive than (infinitary) conditional equations [Tar86b], in the sense that initial models of such specifications are not guaranteed to exist. To illustrate the problem, the following example shows what can go wrong when the language of axioms is extended to permit disjunctions of equations.

Example 2.7.11. Consider the following specification:

$$\begin{array}{l}\textbf{spec}\ \textsc{Status} = \textbf{sorts}\ \mathit{status}\\ \qquad\qquad\quad \textbf{ops}\ \ \mathit{single}:\mathit{status}\\ \qquad\qquad\qquad\quad \mathit{married}:\mathit{status}\\ \qquad\qquad\qquad\quad \mathit{widowed}:\mathit{status}\\ \qquad\qquad\quad \bullet\ \mathit{widowed}=\mathit{single}\vee\mathit{widowed}=\mathit{married}\end{array}$$

where disjunction of equations has the obvious interpretation. There are three kinds of algebras in $\mathit{Mod}[\textsc{Status}]$:

1. Those satisfying $\mathit{single}=\mathit{widowed}=\mathit{married}$.
2. Those satisfying $\mathit{single}=\mathit{widowed}\neq\mathit{married}$.
3. Those satisfying $\mathit{single}\neq\mathit{widowed}=\mathit{married}$.

None of these is an initial model of STATUS: there are no homomorphisms from algebras in the first class to algebras in either of the other two classes, and no homomorphisms in either direction between algebras in the second and third classes. □

In contrast, reachable semantics can be used for specifications with axioms of any form (once a definition of satisfaction of such axioms by algebras has been given, of course). Such flexibility is a distinct advantage of this approach.

Another alternative to initial semantics deserves brief mention.

Definition 2.7.12 (Final semantics). Let $\Sigma=\langle S,\Omega\rangle$ be a signature such that $\mathit{bool}\in S$ and $\mathit{true}:\mathit{bool}$ and $\mathit{false}:\mathit{bool}$ are in Ω. A Σ-algebra $A\in\mathit{RMod}[\langle\Sigma,\Phi\rangle]$ is a *final* (or *terminal*) *model of* $\langle\Sigma,\Phi\rangle$ if for every $B\in\mathit{RMod}[\langle\Sigma,\Phi\rangle]$ there is a unique Σ-homomorphism $h:B\to A$. Taking $\langle\Sigma,\Phi\rangle$ to denote the class of its final models is called taking its *final semantics*. □

As in the case of initial semantics, the final models of a presentation form an isomorphism class. Recall that a model of a presentation is initial iff it contains no junk and no confusion (Exercise 2.5.19). We can give a similar characterisation of final models as the models containing no junk and *maximal confusion*: a final model A satisfies as many ground equations as possible, subject to the restriction that $A\not\models\forall\varnothing\bullet\mathit{true}=\mathit{false}$ (imposed on all reachable standard models).

Example 2.7.13. Recall the specification SETNAT from Example 2.7.7, and the classification of models of SETNAT according to the commutativity and idempotence of *add*. The final models of SETNAT are in the class of "set-like" algebras, in which *add* is both commutative and idempotent. (**Exercise:** Why?) □

Not all presentations with equational axioms have final models, but it is possible to give conditions on the form of presentations that guarantee the existence of final models [BDP$^+$79].

Exercise 2.7.14. Find a variation on the specification STATUS in Example 2.7.11 that has no final models. □

When reachable or final semantics of presentations is used with equational or conditional equational axioms, sometimes more operations are required in specifications than in the case of initial semantics. These additional operations are needed to provide ways of "observing" values of sorts other than *bool*, in order to avoid degenerate models. For example, the presence of the operation *eq* in Example 2.7.7 ensures that $succ^m(0) = succ^n(0)$ only if $m = n$ in all models that do not satisfy $\forall \varnothing \bullet true = false$; it would not be needed if we were interested only in the initial models of SETNAT. Such operations are not required if inequations are allowed as axioms.

Exercise 2.7.15. Recall the presentation NAT given in Exercise 2.5.4. Augment this with the sort *bool* and constants $true, false : bool$ (to make reachable and final semantics applicable), and show that final models of the resulting specification have a single value of sort *nat*. Add an operation $even : nat \to bool$, with the following axioms:

$\forall \varnothing \bullet even(0) = true$
$\forall \varnothing \bullet even(succ(0)) = false$
$\forall n{:}nat \bullet even(succ(succ(n))) = even(n)$

Show that final models of the resulting specification have exactly two values of sort *nat*. Replace *even* with $\_\_ \leq \_\_ : nat \times nat \to bool$, with appropriate axioms, and show that final models of the resulting specification satisfy $succ^m(0) = succ^n(0)$ iff $m = n$. (We have already seen that this is the case if $eq : nat \times nat \to bool$ is added in place of $\leq$.) □

Although the inclusion of additional operations tends to make specifications longer, it is not an artificial device. In practice, one would expect each sort to come with an assortment of operations for creating and manipulating values of that sort, so specifications such as NAT are less natural than NAT augmented with operations like $\leq$ and/or *eq*.

2.7.3 Dealing with partial functions: error algebras

An obvious inadequacy of the framework(s) presented above stems from the use of *total* functions in algebras to interpret the operation names in a signature. Since partial functions are not at all uncommon in Computer Science applications — a very simple example being the predecessor function $pred : nat \to nat$, which is undefined on 0 — a great deal of work has gone into ways of lifting this restriction. Three main approaches are discussed below:

Error algebras (this subsection): Predecessor is regarded as a total function, with $pred(0)$ specified to yield an *error* value.

Partial algebras (Section 2.7.4): Predecessor is regarded as a partial function.

Order-sorted algebras (Section 2.7.5): Predecessor is regarded as a total function on a sub-domain that excludes the value 0.

A fourth approach is to use ordinary (total) algebras, leaving the value of $pred(0)$ unspecified. This is more an attempt to avoid the issue than a solution, and it is workable only in frameworks that deal adequately with non-sufficiently-complete definitions; see Exercises 2.5.20, 2.7.9, and 5.1.11.

The most obvious way of adding error values to algebras does not work, as the following example demonstrates.

Example 2.7.16. Consider the following specification of the natural numbers, where $pred(0)$ is specified to yield an error:

spec NATPRED = **sorts** *nat*
ops $0 : nat$
$succ : nat \to nat$
$pred : nat \to nat$
$error : nat$
$\_\_ + \_\_ : nat \times nat \to nat$
$\_\_ \times \_\_ : nat \times nat \to nat$
$\forall m, n : nat$
- $pred(succ(n)) = n$
- $pred(0) = error$
- $0 + n = n$
- $succ(m) + n = succ(m + n)$
- $0 \times n = 0$
- $succ(m) \times n = (m \times n) + n$

Initial models of NATPRED will have many "non-standard" values of sort *nat*, in addition to the intended one (*error*). For example, the axioms of NATPRED do not force the ground terms $pred(error)$ and $pred(error) + 0$ to be equal to any "normal" value, or to *error*. (**Exercise:** Give an initial model of NATPRED.) A possible solution to this is to add axioms that collapse these non-standard values to a single point:

spec NATPRED = **sorts** *nat*
ops ...
$\forall m, n : nat$
- ...
- $succ(error) = error$
- $pred(error) = error$
- $error + n = error$
- $n + error = error$
- $error \times n = error$
- $n \times error = error$

Unfortunately, NATPRED now has only trivial models: $error = 0 \times error = 0$ and so $error = succ(error) = succ(0)$, $error = succ(error) = succ(succ(0))$, etc. □

The above example suggests that a more delicate treatment is required. A number of approaches have been proposed; here we follow [GDLE84], which is fairly powerful without sacrificing simplicity and elegance. The main ideas of this approach are:

- Error values are distinguished from non-error ("OK") values.
- In an *error signature*, operations that may produce errors when given OK arguments (*unsafe* operations) are distinguished from those that always preserve OK-ness (*safe* operations).
- In an *error algebra*, each carrier is partitioned into an error part and an OK part. Safe operations are required to produce OK results for OK arguments, and homomorphisms are required to preserve OK-ness.
- In equations, variables that can take OK values only (*safe* variables) are distinguished from variables that can take any value (*unsafe* variables). Assignments of values to variables are required to map safe variables to OK values.

Definition 2.7.17 (Error signature). An *error signature* is a triple $\Sigma = \langle S, \Omega, safe \rangle$ where:

- $\langle S, \Omega \rangle$ is an ordinary signature; and
- *safe* is an $S^* \times S$-sorted set of functions $\langle safe_{w,s} : \Omega_{w,s} \to \{tt, ff\} \rangle_{w \in S^*, s \in S}$.

An operation $f : s_1 \times \cdots \times s_n \to s$ in Σ is *safe* if $safe_{s_1 \ldots s_n, s}(f) = tt$; otherwise it is *unsafe*. □

Example 2.7.16 (revisited). An appropriate error signature for NATPRED would be:

$$
\begin{array}{lll}
\Sigma\text{NATPRED} = \textbf{sorts } nat & & \\
\quad \textbf{ops} \quad 0 : nat & & \\
\qquad\quad succ : nat \to nat & & \\
\qquad\quad pred : nat \to nat & : unsafe & \\
\qquad\quad error : nat & : unsafe & \\
\qquad\quad \_\_ + \_\_ : nat \times nat \to nat & & \\
\qquad\quad \_\_ \times \_\_ : nat \times nat \to nat & &
\end{array}
$$

Obviously, *error* is unsafe, and *pred* is unsafe since it produces an error when applied to 0; all the remaining operations are safe. (By convention, the safe operations are those that are not explicitly marked as unsafe.) □

In the rest of this section, let $\Sigma = \langle S, \Omega, safe \rangle$ be an error signature.

Definition 2.7.18 (Error algebra). An *error Σ-algebra A* consists of:

- an ordinary Σ-algebra A; and
- an S-sorted set of functions $OK = \langle OK_s : |A|_s \to \{tt, ff\} \rangle_{s \in S}$

such that safe operations preserve OK-ness: for every $f\!:\!s_1\times\cdots\times s_n\to s$ in Σ such that $safe_{s_1\ldots s_n,s}(f)=tt$ and $a_1\in|A|_{s_1},\ldots,a_n\in|A|_{s_n}$ such that $OK_{s_1}(a_1)=\cdots=OK_{s_n}(a_n)=tt$, $OK_s((f\!:\!s_1\times\cdots\times s_n\to s)_A(a_1,\ldots,a_n))=tt$. A value $a\in|A|_s$ for $s\in S$ is an *OK value* if $OK_s(a)=tt$; otherwise it is an *error value*. □

We employ the usual notational conventions, e.g. writing f_A in place of $(f\!:\!s_1\times\cdots\times s_n\to s)_A$.

Definition 2.7.19 (Error homomorphism). Let A and B be error Σ-algebras. An *error Σ-homomorphism* $h\!:\!A\to B$ is an S-sorted function $h\!:\!|A|\to|B|$ with the usual homomorphism property (for all $f\!:\!s_1\times\cdots\times s_n\to s$ in Σ and $a_1\in|A|_{s_1},\ldots,a_n\in|A|_{s_n}$, $h_s(f_A(a_1,\ldots,a_n))=f_B(h_{s_1}(a_1),\ldots,h_{s_n}(a_n))$) such that h preserves OK-ness: for every $s\in S$ and $a\in|A|_s$ such that $OK_s(a)=tt$ (in A), $OK_s(h_s(a))=tt$ (in B). □

Definition 2.7.20 (Error variable set). An *error S-sorted variable set* X consists of an S-sorted set X such that $X_s\subseteq\mathscr{X}$ for all $s\in S$, and an S-sorted set of functions $safe=\langle safe_s\!:\!X_s\to\{tt,ff\}\rangle_{s\in S}$. A variable $x\!:\!s$ in X is *safe* if $safe_s(x)=tt$; otherwise it is *unsafe*. An *assignment* of values in an error Σ-algebra A to an error S-sorted variable set X is an S-sorted function $v\!:\!X\to|A|$ assigning OK values to safe variables: for every $x\!:\!s$ in X such that $safe_s(x)=tt$, $OK_s(v_s(x))=tt$. □

Definition 2.7.21 (Error algebra of terms). Let X be an error S-sorted variable set. The *error Σ-algebra $ET_\Sigma(X)$ of terms with variables X* is defined in an analogous way to the ordinary term algebra $T_\Sigma(X)$, with the following partition of the S-sorted set of terms into OK and error values:

> For all sorts $s\in S$ and Σ-terms $t\in|ET_\Sigma(X)|_s$, if t contains an unsafe variable or operation then $OK_s(t)=ff$; otherwise $OK_s(t)=tt$.

We adopt the same notational conventions for terms as before, dropping sort decorations etc. when there is no danger of confusion. Let ET_Σ denote $ET_\Sigma(\varnothing)$. □

The definitions of term evaluation, error equation, satisfaction of an error equation by an error algebra, error presentation, model of an error presentation, semantic consequence, and initial model are analogous to the definitions given earlier in the standard many-sorted algebraic framework (Definitions 1.4.5, 2.1.1, 2.1.2, 2.2.1, 2.2.2, 2.3.6 and 2.5.13 respectively). Because assignments are required to map safe variables to OK values, an error equation may be satisfied by an error algebra even if it is not satisfied when error values are substituted for safe variables.

Exercise 2.7.22. Spell out the details of these definitions. □

As before, every error presentation has an isomorphism class of initial models, and an analogous quotient construction gives an initial model.

Definition 2.7.23 (Congruence generated by a set of equations). Let Φ be a set of error Σ-equations. The Σ-congruence $\equiv_\Phi$ on ET_Σ is defined by $t\equiv_\Phi t'\iff\Phi\models_\Sigma\forall\varnothing\bullet t=t'$ for all $t,t'\in|ET_\Sigma|$. $\equiv_\Phi$ is called the *Σ-congruence generated by Φ*. (NOTE: A Σ-congruence on an error Σ-algebra A is just an ordinary Σ-congruence on the ordinary Σ-algebra underlying A.) □

Definition 2.7.24 (Quotient error algebra). Let A be an error Σ-algebra, and let $\equiv$ be a Σ-congruence on A. The definition of $A/\equiv$, the *quotient error algebra of A modulo $\equiv$*, is analogous to that of the ordinary quotient algebra $A/\equiv$, with the following partition of congruence classes into OK and error values:

For all sorts $s \in S$ and congruence classes $[a]_{\equiv_s} \in |A/\equiv|_s$, if there is some $b \in [a]_{\equiv_s}$ such that $OK_s(b) = tt$ (in A) then $OK_s([a]_{\equiv_s}) = tt$ (in $A/\equiv$); otherwise $OK_s([a]_{\equiv_s}) = ff$. □

Note that if there are both OK and error values in a congruence class, the class is regarded as an OK value in the quotient.

Theorem 2.7.25 (Initial model theorem). *The error Σ-algebra $ET_\Sigma/\equiv_\Phi$ is an initial model of the error presentation $\langle \Sigma, \Phi \rangle$.* □

Exercise 2.7.26. Sketch a proof of Theorem 2.7.25. (HINT: Take inspiration from the proof of Theorem 2.5.14.) □

Exercise 2.7.27. Try to find conditions analogous to "no junk" and "no confusion" that characterise the initial models of an error presentation. □

Example 2.7.16 (revisited). Using the approach outlined above, here is an improved version of the specification NATPRED:

spec NATPRED = **sorts** *nat*
ops 0: *nat*
$succ$: *nat* $\to$ *nat*
$pred$: *nat* $\to$ *nat* : *unsafe*
error: *nat* : *unsafe*
$\_\_ + \_\_$: *nat* $\times$ *nat* $\to$ *nat*
$\_\_ \times \_\_$: *nat* $\times$ *nat* $\to$ *nat*
$\forall m, n$: *nat*
• $pred(succ(n)) = n$
• $pred(0) = error$
• $0 + n = n$
• $succ(m) + n = succ(m + n)$
• $0 \times n = 0$
• $succ(m) \times n = (m \times n) + n$

(By convention, variables in equations are safe unless otherwise indicated.) In initial models of NATPRED, the error values of sort *nat* correspond exactly to "error messages", i.e. ground terms containing at least one occurrence of *error*. These terms can be regarded as recording the sequence of events that took place since the error occured. The record is accurate since the initial models of NATPRED do *not* satisfy equations like $\forall \varnothing \bullet 0 \times error = 0$, in contrast to the initial models of the earlier version. To collapse the error values to a single point without affecting the OK values, axioms can be added as follows:

spec NATPRED = **sorts** *nat*
ops ...
$\forall m, n$: *nat*, *k*: *nat*:*unsafe*
- ...
- $pred(error) = error$
- $succ(error) = error$
- $error + k = error$
- $k + error = error$
- $error \times k = error$
- $k \times error = error$

It is also possible to specify *error recovery* using this approach:

spec NATPRED = **sorts** *nat*
ops ...
$recover$: $nat \to nat$
$\forall m, n$: *nat*, *k*: *nat*:*unsafe*
- ...
- $recover(error) = 0$
- $recover(n) = n$

In initial models of this version of NATPRED, *recover* is the identity on *nat* except that $recover(error)$ gives the OK value 0. □

Although only initial semantics of error presentations has been mentioned above, the alternatives of reachable and final semantics apply as in the standard case. The key points of the standard framework not mentioned here (e.g. analogues to the soundness, completeness and incompleteness theorems) carry over to the present framework as well.

Exercise 2.7.28. Find a definition of error signature morphism which makes the Satisfaction Lemma hold, taking the natural definition of the σ-reduct $A'|_\sigma$ of an error Σ'-algebra A' induced by an error signature morphism $\sigma: \Sigma \to \Sigma'$. □

Although the approach to error specification presented above is quite attractive, there are examples that cannot be treated in this framework.

Exercise 2.7.29. Consider the following specification of *bounded natural numbers*:

spec BOUNDEDNAT = **sorts** *nat*
ops 0: *nat*
$succ$: $nat \to nat$: *unsafe*
overflow: *nat* : *unsafe*
- $succ(succ(succ(succ(succ(succ(0)))))) = overflow$

The intention is to specify a (very) restricted subset of the natural numbers, where an attempt to compute a number larger than 5 results in overflow. Show that an initial model of BOUNDEDNAT will have only one OK value. Change BOUNDEDNAT to make its initial models have six OK values (corresponding to $0, succ(0), \ldots, succ^5(0)$). What if the bound is 2^{32} rather than 5? □

2.7.4 Dealing with partial functions: partial algebras

An obvious way to deal with partial functions is to simply change the definition of algebra to allow operation names to be interpreted as partial functions. But for many of the basic notions in the framework that depend on the definition of algebra, beginning with the concepts of subalgebra and homomorphism, there are several ways to extend the usual definition to the partial case. Choosing a coherent combination of these definitions is a delicate matter. Here we follow the approach of [BW82b].

Throughout this section, let $\Sigma = \langle S, \Omega \rangle$ be a signature.

Definition 2.7.30 (Partial algebra). A *partial Σ-algebra A* is like an ordinary Σ-algebra, except that each $f\colon s_1 \times \cdots \times s_n \to s$ in Σ is interpreted as a *partial* function $(f\colon s_1 \times \cdots \times s_n \to s)_A\colon |A|_{s_1} \times \cdots \times |A|_{s_n} \to |A|_s$. The *(total) Σ-algebra underlying A* is the Σ-algebra $A_\perp$ defined as follows:

- $|A_\perp|_s = |A|_s \uplus \{\perp_s\}$ for every $s \in S$; and
- $(f\colon s_1 \times \cdots \times s_n \to s)_{A_\perp}(a_1, \ldots, a_n) =$
$$\begin{cases} \perp_s & \text{if } a_j = \perp_{s_j} \text{ for some } 1 \le j \le n \\ (f\colon s_1 \times \cdots \times s_n \to s)_A(a_1, \ldots, a_n) & \text{if this is defined} \\ \perp_s & \text{otherwise} \end{cases}$$
for every $f\colon s_1 \times \cdots \times s_n \to s$ and $a_1 \in |A_\perp|_{s_1}, \ldots, a_n \in |A_\perp|_{s_n}$. □

We employ the same notational conventions as before. Note that according to this definition, the value of a constant need not be defined: a constant $c\colon s$ is associated in an algebra A with a partial function $c_A\colon \{\langle\rangle\} \to |A|_s$, where $\{\langle\rangle\}$ is the 0-ary Cartesian product.

Definition 2.7.31 (Homomorphism). Let A and B be partial Σ-algebras. A *weak Σ-homomorphism $h\colon A \to B$* is an S-sorted (total) function $h\colon |A| \to |B|$ such that for all $f\colon s_1 \times \cdots \times s_n \to s$ in Σ and $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$,

$$\text{if } f_A(a_1, \ldots, a_n) \text{ is defined then } \begin{array}{l} f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) \text{ is defined, and} \\ h_s(f_A(a_1, \ldots, a_n)) = f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)). \end{array}$$

If moreover h satisfies the condition

$$\text{if } f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) \text{ is defined then } f_A(a_1, \ldots, a_n) \text{ is defined}$$

then h is called a *strong Σ-homomorphism*. □

Other possibilities would be generated by allowing homomorphisms to be partial functions.

Exercise 2.7.32. Consider a partial Σ-algebra A and its underlying total Σ-algebra $A_\perp$. Given any Σ-congruence $\equiv$ on $A_\perp$, removing all pairs involving $\perp$ yields a *strong Σ-congruence on A*. Check that such strong congruences are exactly kernels of strong Σ-homomorphisms, cf. Exercises 1.3.14 and 1.3.18. Check that strong congruences are equivalence relations that preserve and reflect definedness of operations and are closed under defined operations. Kernels of weak Σ-homomorphisms

are *weak Σ-congruences*: equivalence relations that are closed under defined operations. Spell out these definitions in detail. For any partial Σ-algebra A and weak Σ-congruence $\equiv$ on A, generalise Definition 1.3.15 to define the *quotient of A by* $\equiv$, written $A/\equiv$. Note that an operation is defined in $A/\equiv$ on a tuple of equivalence classes provided that in A it is defined on at least one tuple of their respective elements. Check which of Exercises 1.3.18–1.3.23 carry over. □

Definition 2.7.33 (Term evaluation). Let X be an S-sorted set of variables, let A be a partial Σ-algebra, and let $v: X \to |A|$ be a (total) S-sorted function assigning values in A to variables in X. Since $|A| \subseteq |A_\perp|$, this is an S-sorted function $v_\perp: X \to |A_\perp|$, and by Fact 1.4.4 there is a unique (ordinary) Σ-homomorphism $v^\#_\perp: T_\Sigma(X) \to A_\perp$ which extends $v_\perp$. Let $s \in S$ and let $t \in |T_\Sigma(X)|_s$ be a Σ-term of sort s; the *value of t in A under the valuation v* is $v^\#_\perp(t)$ if $v^\#_\perp(t) \neq \perp_s$, and is undefined otherwise. □

Satisfaction of an equation $\forall X \bullet t = t'$, where the values of t and/or t' may be undefined, can be defined in several different ways. Following [BW82b], we use *strong* equality (also known as *Kleene* equality) whereby the equality holds if (for any assignment of values to variables) the values of t and t' are either both defined and equal, or are both undefined. The usual interpretation of definitional equations in recursive function definitions (see for instance Example 4.1.25 below) makes them hold as strong equations. An alternative is *existential equality* (where $=$ is usually written $\stackrel{e}{=}$), whereby the equality holds only when the values of t and t' are defined and equal. When strong equality is used, there is a need for an additional form of axiom called a *definedness formula*: $\forall X \bullet def(t)$ holds if for any assignment of values to variables, the value of t is defined. These are superfluous with existential equality since $\forall X \bullet def(t)$ holds iff $\forall X \bullet t \stackrel{e}{=} t$ holds.

Exercise 2.7.34. Formalize the definitions of satisfaction of equations (using strong equality) and of definedness formulae. □

Using both equations and definedness formulae as axioms, the definitions of presentation, model of a presentation, semantic consequence, isomorphism, and initial model (with respect to *weak* homomorphisms) are analogous to those given earlier.

Exercise 2.7.35. Spell out the details of these definitions. □

Theorem 2.7.36 (Initial model theorem). *Any presentation $\langle \Sigma, \Phi \rangle$ has an initial model I, characterised by the following properties:*

- *I contains no junk;*
- *I is* minimally defined*, i.e. for all $t \in |T_\Sigma|$, t_I is defined only if $\Phi \models_\Sigma \forall \varnothing \bullet def(t)$; and*
- *I contains no confusion, i.e. for all $t, t' \in |T_\Sigma|_s, s \in S$, t_I and t'_I are defined and equal only if $\Phi \models_\Sigma \forall \varnothing \bullet t = t'$.*

Proof sketch. Let $\Sigma_\perp$ be the signature obtained by adding a constant $\perp_s: s$ to Σ for each sort $s \in S$. Define a congruence $\sim \subseteq |T_{\Sigma_\perp}| \times |T_{\Sigma_\perp}|$ as follows: for $t_1, t_2 \in |T_{\Sigma_\perp}|_s$ for some $s \in S$, $t_1 \sim t_2$ iff any of the following conditions holds:

1. t_1 contains $\bot_{s'}$ and t_2 contains $\bot_{s''}$ for some $s', s'' \in S$;
2. t_1 contains $\bot_{s'}$ for some $s' \in S$, $t_2 \in |T_\Sigma|_s$ (so t_2 does not contain $\bot_{s''}$ for any $s'' \in S$) and $\Phi \not\models def(t_2)$, or vice versa
3. $t_1, t_2 \in |T_\Sigma|_s$, and either $\Phi \not\models def(t_1)$ and $\Phi \not\models def(t_2)$ or $\Phi \models t_1 = t_2$.

I is constructed by taking the quotient of $T_{\Sigma_\bot}$ by $\sim$, and then regarding congruence classes containing the constants $\bot_s$ as undefined values. □

Exercise 2.7.37. Complete the above proof by showing that:

- $\sim$ is a congruence on $T_{\Sigma_\bot}$;
- $I \models \Phi$;
- I is an initial model of $\langle \Sigma, \Phi \rangle$; and
- I has the properties promised in Theorem 2.7.36.

Show that any model of $\langle \Sigma, \Phi \rangle$ satisfying the properties in Theorem 2.7.36 is isomorphic to I and is therefore an initial model of $\langle \Sigma, \Phi \rangle$. □

Exercise 2.7.38. Suppose that we modify Theorem 2.7.36 by replacing the phrase "t_I and t'_I are defined and equal" with "$I \models_\Sigma \forall \varnothing \bullet t = t'$". Give a counterexample showing that this version of the theorem is false. □

Exercise 2.7.39. A partial Σ-algebra $A \in Mod[\langle \Sigma, \Phi \rangle]$ is a *strongly initial model of* $\langle \Sigma, \Phi \rangle$ if for every minimally defined $B \in Mod[\langle \Sigma, \Phi \rangle]$ containing no junk, there is a unique strong Σ-homomorphism $h: A \to B$. Show that I is an initial model of $\langle \Sigma, \Phi \rangle$ iff I is a strongly initial model of $\langle \Sigma, \Phi \rangle$. □

Again, reachable and final semantics are applicable for partial algebras as well as initial semantics, and the key points of the standard framework carry over with appropriate changes (for instance, the equational calculus must be modified to deal with definedness formulae as well as equations).

Example 2.7.16 (revisited). Here is a version of the specification NATPRED in which *pred* is specified to be a partial function:

spec NATPRED = **sorts** *nat*
ops $0: nat$
$succ: nat \to nat$
$pred: nat \to nat$
$\_\_ + \_\_ : nat \times nat \to nat$
$\_\_ \times \_\_ : nat \times nat \to nat$
$\forall m, n: nat$
- $def(0)$
- $def(succ(n))$
- $pred(succ(n)) = n$
- $0 + n = n$
- $succ(m) + n = succ(m + n)$
- $0 \times n = 0$
- $succ(m) \times n = (m \times n) + n$

In initial models of NATPRED, all operations behave as expected, and all are total except for *pred* which is undefined only on 0.

Exercise. Show that $\forall m,n{:}nat \bullet def(m+n)$ and $\forall m,n{:}nat \bullet def(m \times n)$ are consequences of the definedness axioms for 0 and *succ* and the equations defining $+$ and $\times$, in reachable models of NATPRED. You will need to use induction, so first formulate an appropriate induction rule scheme and convince yourself that it is sound.

Exercise. Suppose that the axiom $\forall \varnothing \bullet def(0)$ were removed from NATPRED. Describe the initial models of the resulting presentation. □

2.7.5 Partial functions: order-sorted algebras

Any partial function amounts to a total function on a restricted domain. The idea of *order-sorted algebra* is to avoid partial functions by enabling the domain of each function to be specified exactly. This is done by introducing *subsorts*, which correspond to subsets at the level of values, and requiring operations to behave in an appropriate fashion when applied to a value of a subsort or when expected to deliver a value of a supersort. A number of different approaches to order-sorted algebra have been proposed, and their relative merits are still a matter for debate. Here we follow the approach of [GM92].

Definition 2.7.40 (Order-sorted signature). An *order-sorted signature* is a triple $\Sigma = \langle S, \leq, \Omega \rangle$ where $\langle S, \Omega \rangle$ is an ordinary signature and $\leq$ is a partial order on the set S of sort names, such that whenever $f{:}\, s_1 \times \cdots \times s_n \to s$ and $f{:}\, s'_1 \times \cdots \times s'_n \to s'$ are operations (having the same name and same number of arguments) in Ω and $s_i \leq s'_i$ for all $1 \leq i \leq n$, then $s \leq s'$. When $s \leq s'$ for $s, s' \in S$, we say that s is a *subsort* of s' (or equivalently, s' is a *supersort* of s). The subsort ordering is extended to sequences of sorts of equal length in the usual way: $s_1 \ldots s_n \leq s'_1 \ldots s'_n$ if $s_i \leq s'_i$ for all $1 \leq i \leq n$. □

The restriction on Ω ([GM92] calls this condition *monotonicity*) is a fairly natural one, keeping in mind that the subsort ordering corresponds to subset on the value level: restricting a function to a subset of its domain may diminish, but not enlarge, its codomain. Note that an effect of this restriction is to rule out overloaded constants.

Throughout the rest of this section, let $\Sigma = \langle S, \leq, \Omega \rangle$ be an order-sorted signature, and let $\widehat{\Sigma} = \langle S, \Omega \rangle$ be the (ordinary) signature corresponding to Σ.

Definition 2.7.41 (Order-sorted algebra). An *order-sorted Σ-algebra* A is an ordinary $\widehat{\Sigma}$-algebra, such that:

- for all $s \leq s'$ in Σ, $|A|_s \subseteq |A|_{s'}$; and
- whenever $f{:}\, s_1 \times \cdots \times s_n \to s$ and $f{:}\, s'_1 \times \cdots \times s'_n \to s'$ are operations (having the same name and same number of arguments) in Ω and $s_1 \ldots s_n \leq s'_1 \ldots s'_n$, then the function $(f{:}\, s_1 \times \cdots \times s_n \to s)_A{:}\, |A|_{s_1} \times \cdots \times |A|_{s_n} \to |A|_s$ is the set-theoretic restriction of the function $(f{:}\, s'_1 \times \cdots \times s'_n \to s')_A{:}\, |A|_{s'_1} \times \cdots \times |A|_{s'_n} \to |A|_{s'}$. □

An effect of the second restriction ([GM92] calls this condition *monotonicity* as well) is to avoid ambiguity in the evaluation of terms (see below).

Definition 2.7.42 (Order-sorted homomorphism). Let A and B be order-sorted Σ-algebras. An *order-sorted Σ-homomorphism* $h: A \to B$ is an ordinary $\widehat{\Sigma}$-homomorphism, such that $h_s(a) = h_{s'}(a)$ for all $a \in |A|_s$ whenever $s \leq s'$. When h has an inverse, it is an *order-sorted Σ-isomorphism* and we write $A \cong B$. □

Let X be an S-sorted set (of variables) such that X_s and $X_{s'}$ are disjoint for any $s \neq s'$.

Definition 2.7.43 (Order-sorted term algebra). The *order-sorted Σ-algebra $T_\Sigma(X)$ of terms with variables X* is just like $T_{\widehat{\Sigma}}(X)$, except that for any term $t \in |T_\Sigma(X)|_s$ such that $s \leq s'$, we also have $t \in |T_\Sigma(X)|_{s'}$. Let $T_\Sigma = T_\Sigma(\varnothing)$. □

Exercise 2.7.44. Check that $T_\Sigma(X)$ is an order-sorted Σ-algebra. □

Example 2.7.45. One way of reformulating NATPRED as an order-sorted specification (see below) will involve introducing a sort *nznat* (non-zero natural numbers) such that *nznat* $\leq$ *nat*, with operations 0: *nat* and *succ*: *nat* $\to$ *nznat*. According to the definition of order-sorted term algebra, the term $succ(0)$ has sort *nat* as well as *nznat*, which means that $succ(succ(0))$ is well-formed (and has sort *nat* as well as *nznat*). □

As the above example demonstrates, a given term may appear in more than one carrier of $T_\Sigma(X)$. The following condition on Σ ensures that this does not lead to ambiguity.

Definition 2.7.46 (Regular order-sorted signature). Σ is *regular* if for any $f: s_1 \times \cdots \times s_n \to s$ in Σ and $s_1^* \ldots s_n^* \leq s_1 \ldots s_n$, there is a least $s_1' \ldots s_n' s'$ such that $s_1^* \ldots s_n^* \leq s_1' \ldots s_n'$ and $f: s_1' \times \cdots \times s_n' \to s'$ is in Σ. □

Theorem 2.7.47 (Terms have least sorts). *If Σ is regular, then for every term $t \in |T_\Sigma(X)|$ there is a least sort $s \in S$, written $sort(t)$, such that $t \in |T_\Sigma(X)|_s$.* □

Exercise 2.7.48. Prove Theorem 2.7.47. What happens when X is an *arbitrary* S-sorted set, i.e. if we remove the restriction that X_s and $X_{s'}$ are disjoint for any $s \neq s'$? □

Now the definition of term evaluation is analogous to the usual one.

Fact 2.7.49. *Suppose that Σ is regular. Then, for any order-sorted Σ-algebra A and S-sorted function $v: X \to |A|$, there is exactly one order-sorted Σ-homomorphism $v^\#: T_\Sigma(X) \to A$ which extends v, i.e. such that $v_s^\#(x) = v_s(x)$ for all $s \in S$, $x \in X_s$.* □

Exercise 2.7.50. Define term evaluation. □

Definition 2.7.51 (Order-sorted equation; satisfaction). Suppose that Σ is regular, and let the equivalence relation $\equiv$ be the symmetric transitive closure of $\leq$. *Order-sorted Σ-equations* $\forall X \bullet t = t'$ are as usual, except that we require $sort(t) \equiv sort(t')$ (in other words, $sort(t)$ and $sort(t')$ are in the same *connected component* of $\langle S, \leq \rangle$) instead of $sort(t) = sort(t')$. An order-sorted Σ-algebra A *satisfies* an order-sorted Σ-equation $\forall X \bullet t = t'$, written $A \models_\Sigma \forall X \bullet t = t'$, if the value of t in $|A|_{sort(t)}$ and the value of t' in $|A|_{sort(t')}$ coincide, for every S-sorted function $v : X \to |A|$. □

A problem with this definition is that satisfaction of order-sorted Σ-equations is not preserved by order-sorted Σ-isomorphisms (compare Exercise 2.1.5). The following condition on Σ ensures that this anomaly does not arise.

Definition 2.7.52 (Coherent order-sorted signature). $\langle S, \leq \rangle$ is *filtered* if for any $s, s' \in S$ there is some $s'' \in S$ such that $s \leq s''$ and $s' \leq s''$. $\langle S, \leq \rangle$ is *locally filtered* if each of its connected components is filtered. Σ is *coherent* if $\langle S, \leq \rangle$ is locally filtered and Σ is regular. □

Exercise 2.7.53. Find Σ, A, B and φ such that Σ is regular, $A \models_\Sigma \varphi$ and $A \cong B$ but $B \not\models_\Sigma \varphi$. Show that if Σ is coherent then this is impossible. □

The definitions of order-sorted presentation, model of an order-sorted presentation, semantic consequence, and initial model are analogous to those given earlier. For every order-sorted presentation $\langle \Sigma, \Phi \rangle$ such that Σ is coherent, an initial model may be constructed as a quotient of T_Σ [GM92]. There is a version of the equational calculus that is sound and complete for coherent signatures [GM92], and the use of term rewriting for proof as discussed in Section 2.6 is sound, provided that each rewrite rule $t \to t'$ is *sort-decreasing*, i.e. $sort(t') \leq sort(t)$ [KKM88].

Example 2.7.16 (revisited). Here is a version of the specification NATPRED in which *pred* is specified to be a total function on the non-zero natural numbers:

spec NATPRED = **sorts** $nznat \leq nat$
ops $0 : nat$
$succ : nat \to nznat$
$pred : nznat \to nat$
$\_\_ + \_\_ : nat \times nat \to nat$
$\_\_ \times \_\_ : nat \times nat \to nat$
$\forall m, n : nat$
- $pred(succ(n)) = n$
- $0 + n = n$
- $succ(m) + n = succ(m + n)$
- $0 \times n = 0$
- $succ(m) \times n = (m \times n) + n$

In this version of NATPRED, there are terms that are not well-formed in spite of the fact that each operator application seems to be to a value in its domain. For example, consider the following "term":

$pred(succ(0) + succ(0))$.

According to the signature of NATPRED, $succ(0) + succ(0)$ is a term of sort *nat*; it is not a term of sort *nznat* in spite of the fact that its value is non-zero. In the term algebra, *pred* applies only to terms of sort *nznat*, thus the application of *pred* to $succ(0) + succ(0)$ is not defined. One way of getting around this problem might be to add additional operators to the signature of NATPRED:

spec NATPRED = **sorts** $nznat \leq nat$
ops ...
$\_\_ + \_\_ : nznat \times nat \to nznat$
$\_\_ + \_\_ : nat \times nznat \to nznat$
$\_\_ \times \_\_ : nznat \times nznat \to nznat$
...

Then $succ(0) + succ(0)$ is a term of sort *nznat*, as desired. Unfortunately, this signature is not regular. (**Exercise:** Why not? What can be done to make it regular?)

An alternative is to use a so-called *retract*, an additional operation for converting from a sort to one of its subsorts:

spec NATPRED = **sorts** $nznat \leq nat$
ops ...
$r : nat \to nznat$
$\forall m, n : nat, k : nznat$
• ...
• $r(n) = n$

Now, the term $pred(r(succ(0) + succ(0)))$ is well-formed, and is equal to $succ(0)$ in all models of NATPRED. In the words of [GM92], inserting the retract r into $pred(r(succ(0) + succ(0)))$ gives it "the benefit of the doubt", and the term is "vindicated" by the fact that it is equal to a term that does not contain r. The term $pred(r(0))$ is also well-formed, but in the initial model of NATPRED this term is equal only to other terms containing the retract r, and can thus be regarded as an error message. The use of retracts (which can be inserted automatically) is well-behaved under certain conditions on order-sorted presentations [GM92].

Another version of NATPRED is obtained by using an *error supersort* for the codomain of *pred* rather than a subsort for its domain:

spec NATPRED = **sorts** $nat \leq nat?$
ops $0 : nat$
$succ : nat \to nat$
$pred : nat \to nat?$
$\_\_ + \_\_ : nat \times nat \to nat$
$\_\_ \times \_\_ : nat \times nat \to nat$
$\forall m, n : nat$
• $pred(succ(n)) = n$
• $0 + n = n$
• $succ(m) + n = succ(m + n)$
• $0 \times n = 0$
• $succ(m) \times n = (m \times n) + n$

The sort *nat*? may be thought of as *nat* extended by the addition of an error value corresponding to $pred(0)$.

Here we have the same problem with ill-formed terms as before; an example is the term $succ(pred(succ(0)))$. Again, retracts solve the problem. In this case, the required retract is the operation $r : nat? \to nat$, defined by the axiom $\forall n{:}nat \bullet r(n) = n$. □

Exercise 2.7.54. Try to view the error algebra approach presented in Section 2.7.3 as a special case of order-sorted algebra. □

2.7.6 Other options

The previous sections have mentioned only a few of the ways in which the standard framework can be improved to make it more suitable for particular kinds of applications. A great many other variations are possible; a few of these are sketched below.

Example 2.7.55 (First-order predicate logic). Signatures may be modified to enable them to include (typed) *predicate names* in addition to operation names, e.g. $\_\_ \leq \_\_ : nat \times nat$. Atomic formulae are then formed by applying predicates to terms; in *first-order predicate logic with equality*, the predicate $\_\_ = \_\_ : s \times s$ is implicitly available for any sort s. Formulae are built from atomic formulae using logical connectives and quantifiers. Algebras are modified to include relations on their carriers to interpret predicate names; the interpretation of the built-in equality predicate (if available) may be forced to be the underlying equality on values, or it may merely be required to be a congruence relation. Homomorphisms are required to respect predicates as well as operations. The satisfaction of a *sentence* (a formula without free variables) by an algebra is as usual in first-order logic. See Example 4.1.12 for details of the version of first-order predicate logic with equality we will use. Presentations involving predicates and first-order axioms are appropriate for the specification of programs in *logic programming languages* such as Prolog, where the Horn clause fragment of first-order logic is used for writing the programs

themselves. Note that such presentations may have no models at all, but even if they have some models, they may have no initial models (see Example 2.7.11) or no final models (see Exercise 2.7.14), or even no reachable models. (**Exercise:** Give a specification with first-order axioms having some models but no reachable model.) □

Example 2.7.56 (Higher-order functions). Higher-order functions (taking functions as parameters and/or returning functions as results) can be accommodated by interpreting certain sort names as (subsets of) function spaces. Given a set S of (base) sorts, let $S^{\to}$ be the closure of S under formation of function types: $S^{\to}$ is the smallest set such that $S \subseteq S^{\to}$ and for all $s_1,\ldots,s_n,s \in S^{\to}$, $s_1 \times \cdots \times s_n \to s \in S^{\to}$. Then a *higher-order signature* Σ is a pair $\langle S,\Omega\rangle$ where Ω is an $S^{\to}$-indexed set of operation names. This determines an ordinary signature $\Sigma^{\to}$ comprised of the sort names $S^{\to}$ and the operation names in Ω together with operation names $\mathit{apply}\colon (s_1 \times \cdots \times s_n \to s) \times s_1 \times \cdots \times s_n \to s$ for every $s_1,\ldots,s_n,s \in S^{\to}$. Note that, except for the various instances of *apply*, all the operations in $\Sigma^{\to}$ are constants, albeit possibly of "functional" sort. A *higher-order Σ-algebra* is just an ordinary (total) $\Sigma^{\to}$-algebra, and analogously for the definitions of higher-order Σ-homomorphism, reachable higher-order Σ-algebra, higher-order Σ-term, higher-order Σ-equation, satisfaction of a higher-order Σ-equation by a higher-order Σ-algebra, and higher-order presentation. A higher-order Σ-algebra A is *extensional* if for all sorts $s_1 \times \cdots \times s_n \to s \in S^{\to}$ and values $f,g \in |A|_{s_1\times\cdots\times s_n\to s}$, $f = g$ whenever $\mathit{apply}_A(f,a_1,\ldots,a_n) = \mathit{apply}_A(g,a_1,\ldots,a_n)$ for all $a_1 \in |A|_{s_1},\ldots,a_n \in |A|_{s_n}$. In an extensional algebra A, every carrier $|A|_{s_1\times\cdots\times s_n\to s}$ is isomorphic to a subset of the function space $|A|_{s_1} \times \cdots \times |A|_{s_n} \to |A|_s$. A higher-order Σ-algebra A is a *model* of a presentation $\langle\Sigma,\Phi\rangle$ if $A \models_\Sigma \Phi$, A is extensional, and A is reachable. The reachability requirement (no junk) means that $|A|_{s_1\times\cdots\times s_n\to s}$ will almost never be the full function space $|A|_{s_1} \times \cdots \times |A|_{s_n} \to |A|_s$: only the functions that are denotable by ground terms will be present in $|A|_{s_1\times\cdots\times s_n\to s}$. Higher-order (equational) presentations always have initial models [MTW88]. □

Example 2.7.57 (Polymorphic types). Programming languages such as Standard ML [Pau96] can be used to define *polymorphic types* such as α *list* (instances of which include *bool list* and (*bool list*) *list*) and *polymorphic values* such as $\mathit{head}\colon\forall\alpha\bullet\alpha\,\mathit{list}\to\alpha$ (which is then applicable to values of types such as *bool list* and (*bool list*) *list*). To specify such types and functions, signatures are modified to contain *type constructors* in place of sort names; for example, *list* is a unary type constructor and *bool* is a nullary type constructor. Terms built using these type constructors and *type variables* (such as α above) are the *polymorphic types* of the signature. The set Ω of operation names is then indexed by non-empty sequences of polymorphic types, where $f \in \Omega_{t_1\ldots t_n,t}$ means $f\colon\forall FV(t_1)\cup\ldots\cup FV(t_n)\cup FV(t)\bullet t_1\times\cdots\times t_n\to t$. There are various choices for algebras over such signatures. Perhaps the most straightforward choice is to require each algebra A to incorporate a (single-sorted) *algebra of carriers* $\mathit{Carr}(A)$, having sets interpreting types as values and an operation to interpret each type constructor. Then, for each operation $f \in \Omega_{t_1\ldots t_n,t}$ and for each instantiation of type variables $i\colon V \to |\mathit{Carr}(A)|$, A has to provide a function $f_{A,i}\colon i^{\#}(t_1)\times\cdots\times i^{\#}(t_n)\to i^{\#}(t)$. Various conditions may be imposed to ensure that

the interpretation of polymorphic operations is *parametric* in the sense of [Str67], by requiring $f_{A,i}$ and $f_{A,i'}$ to be appropriately related for different type variable instantiations i, i'. Another choice would be to interpret each type as the set of equivalence classes of a *partial equivalence relation* on a model of the untyped λ-calculus [BC88]. Axioms contain (universal) quantifiers for type variables in addition to quantifiers for ordinary variables, as in System F [Gir89]; alternatively, type variable quantification may be left implicit, as in Extended ML [KST97]. □

Example 2.7.58 (Non-deterministic functions). Non-deterministic functions may be handled by interpreting operation names in algebras as relations, or equivalently as set-valued functions. Homomorphisms are required to preserve possible values of functions: for any homomorphism $h: A \to B$ and operation $f: s_1 \times \cdots \times s_n \to s$, if a is a possible value of $f_A(a_1, \ldots, a_n)$ then $h_s(a)$ is a possible value of $f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$. Universally quantified inclusions between sets of possible values may be used as axioms: $t \subseteq t'$ means that every possible value of t is a possible value of t'. □

Example 2.7.59 (Recursive definitions). Following [Sco76], partial functions may be specified as least solutions of recursive equations, where "least" is with respect to an ordering on the space of functions of a given type. To accommodate this, we can use *continuous algebras*, i.e. ordinary (total) Σ-algebras with carriers that are complete partially ordered sets (so-called *cpos*) and operation names interpreted as *continuous functions* on these sets. See Example 3.3.14. The "bottom" element $\bot$ of the carrier for a sort, if it exists, represents the completely undefined value of that sort. The order on carriers induces an order on (continuous) functions in the usual fashion. A homomorphism between continuous algebras is required to be continuous as a function between cpos. It is possible to define a language of axioms that allows direct reference to least upper bounds of chains (see Example 4.1.22), and/or to the order relation itself. Such techniques may also be used to specify infinite data types such as *streams*. □

2.8 Bibliographical remarks

Much of the material presented here is well known, at least in its single-sorted version, in universal algebra as a branch of mathematics. Standard references are [Grä79] and [Coh65]. We approach this material from the direction of computer science, see [Wec92] and [MT92], and present the fundamentals of equational specifications as developed in the 1970s [GTW76], [Gut75], [Zil74], see also [EM85] for an extended monograph-style presentation.

The simplest and most limited form of a specification is a "bare" signature, and this is what is used to characterise classes of algebras (program modules) in modularisation systems for programming languages — see e.g. Standard ML [MTHM97], [Pau96], where such characterisations are in fact called signatures.

The first appearance of the Satisfaction Lemma (Lemma 2.1.8) in the algebraic specification literature was in [BG80], echoing the semantic consequences of the definition of (theory) interpretations in logic [End72]. This fundamental link between syntax and semantics will become one of the cornerstones of later development starting in Chapter 4.

One topic that is only touched upon here (see e.g. Theorem 2.2.10) is the expressive power of specifications. See [BT87] for a comprehensive survey of what is known about the expressive power of the framework presented in this chapter. The main theorem is the one mentioned at the beginning of Section 2.7.

We make a distinction between presentations and theories that is not present in some other work. This distinction surfaces in the definition of theory morphisms (Definition 2.3.11). For two presentations (not necessarily theories) $\langle \Sigma, \Phi \rangle$ and $\langle \Sigma', \Phi' \rangle$, [Gan83] takes a signature morphism $\sigma: \Sigma \to \Sigma'$ to be a specification morphism $\sigma: \langle \Sigma, \Phi \rangle \to \langle \Sigma', \Phi' \rangle$ if $\sigma(\Phi) \subseteq \Phi'$. Such a σ is referred to as an "axiom-preserving theory morphism" in [Mes89]. Exercise 2.3.15 shows that this is not equivalent to our definition of theory morphism between the theories presented by those presentations. Another possibility is to require σ to map only the *ground* equations in Φ to equations in $Cl_{\Sigma'}(\Phi')$, as in [Ehr82]. These alternative definitions seem unsatisfactory since they make little or no sense on the level of models, in contrast to the relationship between theory and model levels for theory morphisms given by Proposition 2.3.13. We will later (Definition 5.5.1) define *specification morphisms*, as a generalisation of morphisms between presentations, relying on this relationship.

The many-sorted equational calculus is presented in [GM85] together with a proof that it is sound and complete. This builds on the standard equational calculus [Bir35], but the modifications needed to deal with empty carriers in the many-sorted context came as a surprise at the time. Our choice of rules in Section 2.4 is different from this standard version but the two systems are equivalent (Exercise 2.4.14) and the proofs of soundness and completeness are analogous.

The initial algebra approach to specification (Section 2.5) is the classical one. It originated with the seminal paper [GTW76], and was further developed by Hartmut Ehrig and his group; see [EM85] for a comprehensive account.

Example 2.5.24 and Exercise 2.5.25 point at useful ways to make inductive proofs easier by providing derived induction rule schemes, as possible for instance in the logics of Larch [GH93] and CASL [Mos04] and their proof support systems (LP [GG89] and HETS [MML07], respectively), see also Chapter 6 of [Far92].

The proof of the incompleteness theorem for initial semantics (Theorem 2.5.26) from [MS85] follows [Nou81] where it was used to show that the equational calculus with a specific induction rule scheme is not complete. An alternative to adding induction rules to the equational calculus is to restrict attention to so-called ω-complete presentations; these are presentations $\langle \Sigma, \Phi \rangle$ for which the equational calculus itself yields all of the Σ-equations that hold in initial models of $\langle \Sigma, \Phi \rangle$ [Hee86]. Then the problem becomes one of finding an ω-complete presentation corresponding to a given presentation. By the incompleteness theorem, this is not always possible.

There is a substantial body of theory on term rewriting systems; Section 2.6 is only the tip of the iceberg. For much more on the topic, and for the details of the Knuth-Bendix completion algorithm [KB70] that have been omitted in Section 2.6, see [DJ90], [Klo92], [BN98], [Kir99] and [Ter03]. See [KM87] or [DJ90] for a discussion of proof by consistency, which originated with [Mus80]. Like most work in this area, all these restrict attention to the single-sorted case. See [EM85] for a treatment of the many-sorted case, up to the soundness and completeness theorems for conversion, without our simplifying assumption (cf. Exercise 2.6.11).

In the case of reachable and final semantics, it is usual to look at reachable or final *extensions* of algebras (alternative terminology: hierarchical specifications), rather than at the reachable or final interpretation of a completed specification. See [BDP$^+$79] or [WB82] for reachable semantics, and [GGM76] or [Wan79] for final semantics. Under appropriate conditions, the reachable models of a presentation form a complete lattice, with the initial model at one extreme and the final model at the other; see [GGM76] and [BWP84]. For such hierarchical specifications, an incompleteness theorem that is even stronger than Theorem 2.5.26 may be proved: no sound proof system can derive all *ground* equational consequences of such specifications, see [MS85].

The first attempt to specify errors by distinguishing error values from OK values was [Gog78]. More details of the approach outlined in Section 2.7.3 can be found in [GDLE84]. The final semantics of error presentations is discussed in [Gog85]. See [BBC86] for an alternative approach which is able to deal with examples like the one discussed in Exercise 2.7.29.

More details of the approach to partial algebras outlined in Section 2.7.4 can be found in [BW82b]. Weak Σ-homomorphisms are called total Σ-homomorphisms there. Alternative approaches to the specification of partial algebras are presented in [Rei87] and [Kre87], and more recently [Mos04]. See [Bur86] for a comprehensive analysis of the various alternative definitions of the basic notions.

See [GM92], further refined in [Mes09], for more on the approach to order-sorted algebra in Section 2.7.5. Alternative approaches include [Gog84], [Poi90] and [Smo86] which is sometimes referred to as "universal" order-sorted algebra to distinguish it from "overloaded" order-sorted algebra as presented here. A universal order-sorted algebra contains a single universe of values, where a sort corresponds to a subset of the universe and each operation name identifies a (single) function on the universe. A compromise is in rewriting logic [Mes92] as implemented in Maude [CDE$^+$02]. See [GD94a] and [Mos93] for surveys comparing the different approaches. [GD94a] discusses how some of the definitions and results in Section 2.7.5 can be generalised by dropping or weakening the monotonicity requirements on order-sorted signatures and order-sorted algebras. Yet a different approach to subsorting is taken in CASL [Mos04] where subsort coercions may be arbitrary injective functions rather than merely inclusions.

First-order predicate logic has been used as a framework for algebraic specification in various approaches, see for instance CIP-L [BBB$^+$85] and CASL [Mos04]. See [Poi86], [MTW88], [Mei92] and [Qia93] for different approaches to the algebraic specification of higher-order functions. Frameworks that cater for the spec-

ification of polymorphic types and functions are described in [MSS90], [Mos89] and [KST97]. See [Nip86] for more on algebras with non-deterministic operations; for a different approach using relation algebra, see [BS93]. See [WM97] for a comprehensive overview. Soundness and completeness of term rewriting for non-deterministic specifications is studied in [Hus92]. Continuous algebras and the use of Scott-style domain-theoretic techniques in algebraic specification were first discussed in [GTWW77]. See [Sch86] or [GS90] for much more on domain theory itself. Although these and other extensions to the standard framework have been explored separately, the few attempts that have been made to combine such extensions (see e.g. [AC89] and [Mos04]) have tended to reveal new problems.

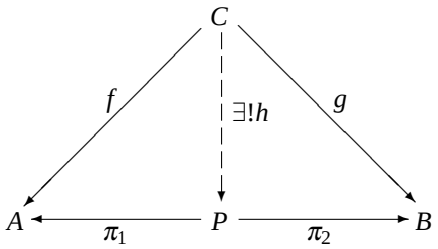
Chapter 3
Category theory

One of the main purposes of this book is to present a general, abstract theory of specifications, which is independent from the exact details of the semantic structures (algebras) used to model particular aspects of program behaviour. Appropriate mathematical tools are required to support the development of such a theory. The basics of category theory provide us with just what we need: a simple, yet powerful language that allows definitions and results to be formulated at a sufficiently general, abstract level.

The most fundamental "categorical dogma" is that for many purposes it does not really matter exactly what the objects we study are; more important are their mutual relationships. Hence, objects should never be considered on their own, they should always come equipped with an appropriate notion of a *morphism* between them. In many typical examples, the objects are sets with some additional structure imposed on them, and their morphisms are maps that preserve this structure. "Categorical dogma" states that the interesting properties of objects may be formulated purely in terms of morphisms, without referring to the internal structure of objects at all. As a very simple example, consider the following two definitions.

Definition. Given two sets A and B, the *Cartesian product* of A and B is the set $A \times B$ that consists of all the pairs of elements from A and B, respectively: $A \times B = \{\langle a, b\rangle \mid a \in A, b \in B\}$ □

Definition. Given two sets A and B, a *product* of A and B is a set P together with two functions $\pi_1 : P \to A$ and $\pi_2 : P \to B$ such that for any set C with functions $f : C \to A$ and $g : C \to B$ there exists a unique function $h : C \to P$ such that $h;\pi_1 = f$ and $h;\pi_2 = g$.



□

It is easy to see that the Cartesian product of any two sets is a product in the sense of the latter definition, where the functions π_1 and π_2 are the projections on the first and second components respectively (HINT: Define $h: C \to A \times B$ by $h(c) = \langle f(c), g(c) \rangle$ for all $c \in C$). Moreover, although a product P of two sets A and B does not have to be their Cartesian product $A \times B$ since the elements of P do not have to be pairs of objects from A and B, P is always isomorphic to $A \times B$ (there is a one-to-one correspondence between elements of P and of $A \times B$). Thus, the two definitions may be viewed as equivalent for many purposes.

The reader may feel that the former definition (of the Cartesian product) is far simpler than the latter (of a product). Indeed, to most of us, brought up to consider set-theoretic concepts as the basis of all mathematics, this is in fact the case. However, the former definition suffers from a serious deficiency: it is formulated in terms of elements and the membership relation for sets (which constitute the specific internal structure of sets). Consequently, it is very specifically oriented towards defining the Cartesian product of sets and of sets only. If we now wanted to define the Cartesian product of, say, algebras (cf. Definition 1.2.9) we would have to reformulate this definition substantially (in this case, by adding definitions of operations for product algebras). To define the Cartesian product of structures of yet another kind, yet another different version of this definition would have to be explicitly stated. It is obviously desirable to avoid such repetition of the same story for different specific kinds of objects whenever possible.

The latter definition (of a product) is quite different from this point of view. It does not make reference to the internal structure of sets at all; it defines a product of two sets entirely in terms of its relationships with these sets and with other sets. To obtain a definition of a product of two algebras, it is enough to replace "set" by "algebra" and "function" by "homomorphism". The same would apply to other kinds of structures, as long as there is an appropriate notion of a morphism between them.

The conclusion we draw from this example is that, first of all, objects of any kind should be considered together with an appropriate notion of a morphism between them, and then, that the structure imposed on the collection of objects by these morphisms should be exploited to formulate definitions at an appropriate level of generality and abstraction.

Let us have a look at another example:

Definition. A function $f: A \to B$ is *surjective* if for every $b \in B$ there exists $a \in A$ such that $b = f(a)$. □

Definition. A function $f: A \to B$ is an *epimorphism* if for any two functions $g, g': B \to C$, $f;g = f;g'$ implies $g = g'$. □

Definition. A function $f: A \to B$ is a *retraction* if there exists a function $g: B \to A$ such that $g;f = id_B$. □

All the three definitions above are equivalent: a function is surjective if and only if it is an epimorphism, if and only if it is a retraction. As with the previous example,

one may argue that the first of these definitions is very much specific to sets, and so not abstract and not general enough. The two other definitions lack this deficiency: they do not refer to the internal structure of sets, but use functions (set morphisms) to define the concept. However, the two definitions when applied to other kinds of objects (and their morphisms) may well turn out not to be equivalent. We cannot say that one of them is "right" and the other is "wrong"; they simply incorporate different aspects of what for sets is the property of "being surjective". The lesson to draw from this is that one has to be cautious when generalising a certain property to a more abstract setting. An attempt to formulate a definition at a more general level should provide us with a better understanding of the essence of the property being defined; it may well turn out, however, that there is more than one essence in it, giving several non-equivalent ways to reformulate the definition in a more abstract way.

Finding an adequate generalisation is not always easy. Sometimes even very simple notions we are accustomed to viewing as fundamental are difficult to formulate in categorical terms, as they depend in an essential way on the internal structure of the objects under consideration, which is exactly what we want to abstract from. The usual set-theoretic union operation is an example of such a notion.

Once we succeed in providing a more general version of a certain notion, it may be instantiated in many different ways. It is interesting to observe how often an adequate generalisation of an important specific concept leads to interesting instantiations in the context of objects (and morphisms between them) different from the ones we started with. Indeed, interesting instantiations in other contexts may be regarded as a test of the adequacy of the generalisation.

A more wide-ranging polemic on the advantages of category theory presented at a rather intuitive level may be found in [Gog91b].

With these remarks in mind, this chapter introduces the basic concepts and results of category theory. It is not our intention to provide a full-blown introductory text on category theory; although a few concepts are introduced which will not be used elsewhere in this book, we consciously refrain from discussing many important but more involved concepts and results. Our aim in this chapter is to provide a brief but comprehensive overview of the basics of category theory, both in order to make this book self-contained and to provide a handy reference.

3.1 Introducing categories

3.1.1 Categories

Definition 3.1.1 (Category). A *category* **K** consists of:

- a collection $|\mathbf{K}|$ of **K**-*objects*;
- for each $A, B \in |\mathbf{K}|$, a collection $\mathbf{K}(A,B)$ of **K**-*morphisms* from A to B; and

- for each $A,B,C \in |\mathbf{K}|$, a *composition operation*[1] $;:\mathbf{K}(A,B)\times\mathbf{K}(B,C)\to\mathbf{K}(A,C)$

such that:

1. for all $A,B,A',B' \in |\mathbf{K}|$, if $\langle A,B\rangle \neq \langle A',B'\rangle$ then $\mathbf{K}(A,B)\cap\mathbf{K}(A',B') = \varnothing$;
2. (*existence of identities*) for each $A \in |\mathbf{K}|$, there is a morphism $id_A \in \mathbf{K}(A,A)$ such that $id_A;g = g$ for all morphisms $g \in \mathbf{K}(A,B)$ and $f;id_A = f$ for all morphisms $f \in \mathbf{K}(B,A)$; and
3. (*associativity of composition*) for any $f \in \mathbf{K}(A,B)$, $g \in \mathbf{K}(B,C)$ and $h \in \mathbf{K}(C,D)$, $f;(g;h) = (f;g);h$. □

Notation. We refer to *objects* and *morphisms* instead of **K**-objects and **K**-morphisms when **K** is clear from the context. We write $f:A\to B$ (in **K**) for $A,B \in |\mathbf{K}|$, $f \in \mathbf{K}(A,B)$. For any $f:A\to B$, we will refer to A as the *source* or *domain*, and to B as the *target* or *codomain* of f. The collection of all morphisms of **K** will be (ambigously) denoted by **K** as well, i.e., $\mathbf{K} = \bigcup_{A,B\in|\mathbf{K}|}\mathbf{K}(A,B)$. □

The above is just one of several possible equivalent definitions of a category. For example, the identities, the existence of which is required in (2), are sometimes considered as part of the structure of a category.

Exercise 3.1.2. Prove that in any category, identities are unique. □

The notion of a category is very general. Accepting the categorical dogma that objects of any kind come equipped with a notion of morphism between them, it is difficult to think of a collection of objects and accompanying morphisms that do not form a category. Almost always there is a natural operation of morphism composition, which obeys two of the basic requirements above: it has identities and is associative. Perhaps requirement (1), which allows us to unambigously identify the source and target of any morphism, is the most technical and hence least intuitively appealing. But even in cases where the same entity may be viewed as a morphism between different objects, this entity can always be equipped with an explicit indication of the source and target of the morphism (cf. Example 3.1.6), thus satisfying requirement (1).

In the rest of this subsection we give a number of examples of categories. We start with some rather trivial examples, mainly of formal interest, and only then define some more typically considered categories. Further examples, which are often more complex, may be found in the following sections of this chapter (and in later chapters, see e.g. Section 10.3 for somewhat more complex examples).

Example 3.1.3 (Preorder categories). A binary relation $\leq \subseteq X\times X$ is a *preorder on* X if:

- $x \leq x$ for all $x \in X$; and
- $x \leq y \wedge y \leq z \Rightarrow x \leq z$ for all $x,y,z \in X$.

[1] We will use semicolon ; to denote composition of morphisms in any category, just as we used it for composition of functions and homomorphisms in the preceding chapters. Composition will always be written in diagrammatic order: $f;g$ is to be read as "f followed by g".

A *preorder* category is a category that has at most one morphism with any given source and target.

Every preorder $\leq \subseteq X \times X$ gives rise to a preorder category $\mathbf{K}_{\leq}$ where $|\mathbf{K}_{\leq}| = X$ and $\mathbf{K}_{\leq}(x,y)$ has exactly one element if $x \leq y$ and is empty otherwise.

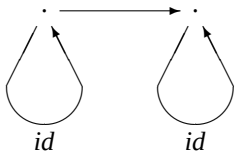
This definition does not identify the category $\mathbf{K}_{\leq}$ unambigously, since different elements may be used as morphisms in $\mathbf{K}_{\leq}(x,y)$ for $x \leq y$. However, we will not worry here about the exact nature of morphisms (nor objects) in a category, and we will treat this and similar definitions below as sufficient. More formally, all categories satisfying the above requirements are isomorphic in the technical sense to be discussed in Section 3.4 (cf. Definition 3.4.68).

Here are some trivial examples of preorder categories:

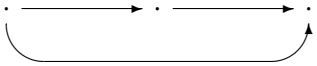
0: (the empty category)

1:

2:

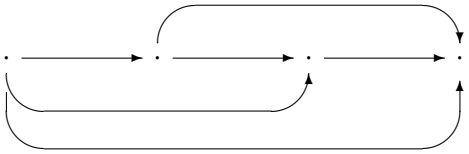


3:



(+ identities)

4:



(+ identities)

⋮ ⋮

Exercise. How many morphisms does **n** have? □

Example 3.1.4 (Discrete category). A category **K** is *discrete* whenever for all $A, B \in |\mathbf{K}|$, $\mathbf{K}(A,B)$ is empty if $A \neq B$ and contains exactly one element (the identity) otherwise.

Any collection of objects X gives rise to a discrete category $\mathbf{K}_X$ where $|\mathbf{K}_X| = X$. □

Example 3.1.5 (Monoid category). A category **K** is a *monoid* if **K** has exactly one object.

A set X together with a function $; : X \times X \to X$ and a distinguished element $id \in X$ is a *monoid* $\langle X, ;, id \rangle$ if $(x;y);z = x;(y;z)$ and $id;x = x;id = x$ for all $x, y, z \in X$. Every monoid $\langle X, ;, id \rangle$ gives rise to a monoid (category) having morphisms X and composition $;$. □

Example 3.1.6 (Set, the category of sets). The category **Set** of sets with functions as morphisms is defined as follows:

Objects of **Set**: sets;
Morphisms of **Set**: functions; however, to ensure that the requirements stated in Definition 3.1.1 are satisfied (disregarding the particular mathematical representation of the concept of a function one uses), we will always consider functions with explicitly given domain and codomain. Thus, a morphism in the category **Set** with source A and target B is a triple $\langle A, f, B \rangle$, where $f : A \to B$ is a function. □

Example 3.1.7 ($\mathbf{Set}^S$, the category of S-sorted sets). For any set S, the category $\mathbf{Set}^S$ of S-sorted sets is defined as follows:

Objects of $\mathbf{Set}^S$: S-sorted sets;
Morphisms of $\mathbf{Set}^S$: S-sorted functions (with explicitly given domain and codomain). □

Example 3.1.8 ($\mathbf{Alg}(\Sigma)$, the category of Σ-algebras). For any signature Σ, the category $\mathbf{Alg}(\Sigma)$ of Σ-algebras is defined as follows:

Objects of $\mathbf{Alg}(\Sigma)$: Σ-algebras;
Morphisms of $\mathbf{Alg}(\Sigma)$: Σ-homomorphisms (with explicitly given domain and codomain). □

Example 3.1.9 (CPO, the category of complete partial orders). The category **CPO** of complete partial orders[2] and continuous functions between them is defined as follows:

Objects of **CPO**: complete partial orders, i.e., partially ordered sets $\langle X, \leq \rangle$ such that any countable chain $x_0 \leq x_1 \leq \ldots$ in $\langle X, \leq \rangle$ has a least upper bound $\bigsqcup_{i \geq 0} x_i$;
Morphisms of **CPO**: continuous functions, i.e., functions that preserve least upper bounds of countable chains. □

Exercise 3.1.10. Complete the above examples by formalising composition in the obvious way. Indicate identities and prove associativity of composition. □

Example 3.1.11 (AlgSig, the category of algebraic signatures). The category **AlgSig** of (algebraic) signatures is defined as follows:

[2] Cpos and continuous functions as defined here are often referred to as ω-cpos and ω-continuous functions, respectively.

Objects of **AlgSig**: signatures;
Morphisms of **AlgSig**: signature morphisms;
Composition in **AlgSig**: for any $\sigma: \Sigma \to \Sigma'$ and $\sigma': \Sigma' \to \Sigma''$, their composition $\sigma;\sigma': \Sigma \to \Sigma''$ is given by $(\sigma;\sigma')_{sorts} = \sigma_{sorts};\sigma'_{sorts}$ and $(\sigma;\sigma')_{ops} = \sigma_{ops};\sigma'_{ops}$, cf. Exercise 1.5.3. □

Exercise 3.1.12 ($\mathbf{AlgSig}^{der}$, the category of signatures with derived morphisms). Recall the concept of a derived signature morphism from Definition 1.5.14. Define the category $\mathbf{AlgSig}^{der}$ of algebraic signatures with derived signature morphisms. Use Exercise 1.5.18 to define composition of derived signature morphisms. □

Example 3.1.13 ($\mathbf{T}_\Sigma$, the category of substitutions over a signature Σ). Recall (cf. Section 1.4) that for any signature $\Sigma = \langle S, \Omega \rangle$ and S-sorted set of variables X, $T_\Sigma(X)$ is the algebra of terms over Σ with variables X. $T_\Sigma(X)$ is characterised up to isomorphism by the property that for any Σ-algebra A, any S-sorted map $v: X \to |A|$ uniquely extends to a Σ-homomorphism $v^\#: T_\Sigma(X) \to A$ (Facts 1.4.4 and 1.4.10).

For any algebraic signature Σ, the category $\mathbf{T}_\Sigma$ of substitutions over Σ is defined as follows (cf. Exercise 1.4.9):

Objects of $\mathbf{T}_\Sigma$: S-sorted sets (of variables);
Morphisms of $\mathbf{T}_\Sigma$: for any sets X and Y, a morphism θ from X to Y is a substitution of terms with variables Y for variables X, i.e., an S-sorted function $\theta: X \to |T_\Sigma(Y)|$;
Composition in $\mathbf{T}_\Sigma$: given any sets X, Y and Z, and morphisms $\theta: X \to Y$ and $\theta': Y \to Z$ in $\mathbf{T}_\Sigma$, i.e., functions $\theta: X \to |T_\Sigma(Y)|$ and $\theta': Y \to |T_\Sigma(Z)|$, their composition $\theta;\theta': X \to Z$ is the function $\theta;\theta': X \to |T_\Sigma(Z)|$ defined by $(\theta;\theta')_s(x) = (\theta')^\#_s(\theta_s(x))$ for all $s \in S$, $x \in X_s$. □

Exercise 3.1.14 ($\mathbf{T}_\Sigma/\Phi$, the category of substitutions over Σ modulo equations Φ). Generalise the above definition of the category of substitutions by considering terms up to an equivalence generated by a set of equations. That is, for any algebraic signature $\Sigma = \langle S, \Omega \rangle$ and set Φ of Σ-equations, for any S-sorted set of variables X define two terms $t_1, t_2 \in |T_\Sigma(X)|_s$ (for any sort $s \in S$) to be equivalent, written $t_1 \equiv t_2$, if $\Phi \vdash_\Sigma \forall X \bullet t_1 = t_2$ (cf. Section 2.4). Now, by analogy with the category of substitutions, define the category $\mathbf{T}_\Sigma/\Phi$ to have S-sorted sets as objects and substitutions modulo Φ as morphisms. A substitution of terms modulo Φ with variables Y for variables X is an S-sorted function $\theta: X \to (|T_\Sigma(Y)|/\equiv)$. Composition in $\mathbf{T}_\Sigma/\Phi$ is defined analogously as in $\mathbf{T}_\Sigma$, by choosing a representative of each of the equivalence classes assigned to variables: given $\theta: X \to (|T_\Sigma(Y)|/\equiv)$ and $\theta': Y \to (|T_\Sigma(Z)|/\equiv)$, $\theta;\theta': X \to (|T_\Sigma(Z)|/\equiv)$ maps any $x \in X$ to $(\theta')^\#(t)$, where $\theta(x) = [t]_\equiv$ (show that the result does not depend on the choice of the representative $t \in \theta(x)$). □

Exercise 3.1.15 ($\mathbf{T}_{\Sigma,\Phi}$, the algebraic $\langle \Sigma, \Phi \rangle$-theory). Building on the definition of the category of substitutions modulo a set of equations sketched above, abstract away from the actual names of variables used in the objects of $\mathbf{T}_\Sigma/\Phi$ by listing them in some particular order, as in derived signatures (cf. Definition 1.5.13). That is, for

any algebraic signature $\Sigma = \langle S, \Omega \rangle$ and set Φ of Σ-equations, define the category $\mathbf{T}_{\Sigma,\Phi}$ with sequences $s_1 \dots s_n \in S^*$ of sort names as objects. A morphism in $\mathbf{T}_{\Sigma,\Phi}$ from $s_1 \dots s_n \in S^*$ to $s'_1 \dots s'_m \in S^*$ is an n-tuple $\langle [t_1]_{\equiv}, \dots, [t_n]_{\equiv} \rangle$ of terms modulo Φ, where the equivalence $\equiv$ is sketched in Exercise 3.1.14 above, and for $i = 1, \dots, n$, $t_i \in |T_\Sigma(I_{s'_1 \dots s'_m})|_{s_i}$, with $I_{s'_1 \dots s'_m} = \{\boxed{1}{:}s'_1, \dots, \boxed{m}{:}s'_m\}$. The composition in $\mathbf{T}_{\Sigma,\Phi}$ is given by substitution on representatives of equivalence classes (the position of a term in a tuple identifies the variable it is to be substituted for). $\mathbf{T}_{\Sigma,\Phi}$ is usually referred to as the *algebraic theory* over Σ generated by Φ.[3] □

3.1.1.1 Foundations

In the above, and in the definition of a category in particular, we have very cautiously used the non-technical term *collection*, and talked of *collections* of objects and morphisms. This allowed us to gloss over the issue of the choice of appropriate set-theoretical foundations for category theory. Even a brief look at the examples above indicates that we could not have been talking here just of *sets* (in the sense of Zermelo-Fraenkel set theory): we want to consider categories like **Set**, where the collection of objects consists of all sets, and so cannot be a set itself. Using *classes* (collections of sets that are possibly too "large" to be sets themselves, as in Bernays-Gödel set theory) might seem more promising, since if we replace the term "collection" by "class" in Definition 3.1.1 then at least examples of categories like **Set** would be covered. However, this is not enough either, since even in this simple presentation of the basics of category theory we will encounter some categories (like **Cat**, the category of "all" categories, and functor categories defined later in this chapter) where objects themselves are proper classes and the collection of objects forms a "conglomerate" (a collection of classes that is too "large" to be a class, cf. [HS73]). We refer to [Bén85] for a careful analysis of the basic requirements imposed on a set theory underlying category theory.

Perhaps the most traditional solution to the problem of set-theoretic foundations for category theory is sketched in [Mac71]. The idea is to work within a hierarchy of *set universes* $\langle \mathbf{U}_n \rangle_{n \geq 0}$, where each universe $\mathbf{U}_n$, $n \geq 0$, is closed under the standard set-theoretic operations, and is an element of the next universe in the hierarchy, $\mathbf{U}_n \in \mathbf{U}_{n+1}$. Then there is a notion of category corresponding to each level of the hierarchy, and one is required to indicate at which level of the hierarchy one is working at any given moment.

However, in our view such pedantry would hide the intuitive appeal of "naive" category theory. We will therefore ignore the issue of set-theoretic foundations for category theory in the sequel, with just one exception: we define what it means for a category to be (locally) small and use this to occasionally warn the reader about potential foundational hazards.

[3] In the literature, the algebraic theory over Σ generated by Φ is often defined with substitutions considered as morphisms in the opposite direction, i.e., as the category $\mathbf{T}^{op}_{\Sigma,\Phi}$ opposite to $\mathbf{T}_{\Sigma,\Phi}$ (cf. Definition 3.1.21 below).

Definition 3.1.16 (Small category). A category $\mathbf{K}$ is *locally small* if for any $A, B \in |\mathbf{K}|$, $\mathbf{K}(A,B)$ is a set (an element of the lowest-level universe $\mathbf{U}_0$); $\mathbf{K}$ is *small* if in addition $|\mathbf{K}|$ is a set as well. □

3.1.2 Constructing categories

In the examples of the previous subsection, each category was constructed "from scratch" by explicitly defining its objects and morphisms and their composition. Category theory also provides numerous ways of modifying a given category to yield a different one, and of putting together two or more categories to obtain a more complicated one. Some of the simplest examples are given in this subsection.

3.1.2.1 Subcategories

Definition 3.1.17 (Subcategory). A category **K1** is a *subcategory* of a category **K2** if $|\mathbf{K1}| \subseteq |\mathbf{K2}|$ and $\mathbf{K1}(A,B) \subseteq \mathbf{K2}(A,B)$ for all objects $A, B \in |\mathbf{K1}|$, with composition and identities in **K1** the same as in **K2**. **K1** is a *full* subcategory of **K2** if additionally $\mathbf{K1}(A,B) = \mathbf{K2}(A,B)$ for all $A, B \in |\mathbf{K1}|$. **K1** is a *wide* subcategory of **K2** if $|\mathbf{K1}| = |\mathbf{K2}|$. □

For any category $\mathbf{K}$, any collection $X \subseteq |\mathbf{K}|$ of objects of $\mathbf{K}$ determines a full subcategory $\mathbf{K}|_X$ of $\mathbf{K}$, defined by $|\mathbf{K}|_X| = X$. Whenever convenient, if $\mathbf{K}$ is evident from the context, we will identify collections $X \subseteq |\mathbf{K}|$ with $\mathbf{K}|_X$.

Example 3.1.18 (FinSet, the category of finite sets). The category **FinSet** of finite sets is defined as follows:

Objects of **FinSet**: finite sets;
Morphisms and composition in **FinSet**: as in **Set**.

FinSet is a full subcategory of **Set**. □

Example 3.1.19. The category of single-sorted signatures is a full subcategory of the category **AlgSig** of (many-sorted) signatures.

The discrete category of sets is a subcategory of the category of sets with inclusions as morphisms, which is a subcategory of the category of sets with injective functions as morphisms, which is a subcategory of **Set**.

For any signature Σ and set Φ of Σ-equations, the class $Mod_\Sigma(\Phi)$ of Σ-algebras that satisfy Φ determines a full subcategory of $\mathbf{Alg}(\Sigma)$, which we denote by $\mathbf{Mod}(\Sigma, \Phi)$. □

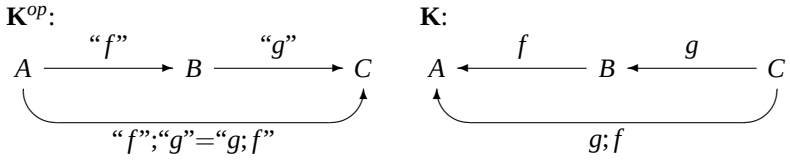
Exercise 3.1.20. Give an example of two categories **K1**, **K2** such that $|\mathbf{K1}| \subseteq |\mathbf{K2}|$, $\mathbf{K1}(A,B) \subseteq \mathbf{K2}(A,B)$ for all objects $A, B \in |\mathbf{K1}|$, with composition in **K1** the same as in **K2**, but such that **K1** is *not* a subcategory of **K2**. □

3.1.2.2 Opposite categories and duality

One of the fundamental theorems of lattice theory (cf. e.g. [DP90]) is the so-called *duality principle*. Any statement in the language of lattice theory has a dual, obtained by systematically replacing greatest lower bounds by least upper bounds and vice versa. The duality principle states that the dual of any theorem of lattice theory is a theorem as well. In a sense, this allows the number of proofs in lattice theory to be cut by half: proving a fact gives its dual "for free". A very similar phenomenon occurs in category theory; in fact, the duality principle of lattice theory may be viewed as a consequence of a more general duality principle of category theory. Replacing greatest lower bounds by least upper bounds and vice versa is generalised here to the process of "reversing morphisms".

Definition 3.1.21 (Opposite category). The *opposite category* of a category $\mathbf{K}$ is the category $\mathbf{K}^{op}$ where:

Objects of $\mathbf{K}^{op}$: $|\mathbf{K}^{op}| = |\mathbf{K}|$;
Morphisms of $\mathbf{K}^{op}$: $\mathbf{K}^{op}(A,B) = \mathbf{K}(B,A)$ for all $A,B \in |\mathbf{K}^{op}|$;
Composition in $\mathbf{K}^{op}$: for $f \in \mathbf{K}^{op}(A,B)$ (i.e., $f \in \mathbf{K}(B,A)$) and $g \in \mathbf{K}^{op}(B,C)$ (i.e., $g \in \mathbf{K}(C,B)$), $f;g \in \mathbf{K}^{op}(A,C)$ is $g;f \in \mathbf{K}(C,A)$.



□

Exercise 3.1.22. Check that:

1. $\mathbf{K}^{op}$ is a category.
2. $(\mathbf{K}^{op})^{op} = \mathbf{K}$.
3. Identities in $\mathbf{K}^{op}$ are the same as in $\mathbf{K}$.

□

If W is a categorical concept (property, statement, ...) then its *dual*, *co-W*, is obtained by reversing all the morphisms in W. This idea may be formalised in two ways. The first is to introduce a formal language of category theory, and then define the operation of forming a dual as an operation on formal statements in this language. The other is to formally interpret *co-W* in a category $\mathbf{K}$ as W in the category $\mathbf{K}^{op}$. Since formalising the language of category theory is beyond the scope of this book (but cf. [Mac71] or [Hat82]), we take the second option here and will rely on an intuitive understanding of duality in the sequel. For example, consider the following property of objects in a category:

$$P(X): \text{ for any object } Y \text{ there is a morphism } f: Y \to X.$$

Then:

$$\textit{co-}P(X): \text{ for any object } Y \text{ there is a morphism } f: X \to Y.$$

Note that indeed *co-*$P(X)$ in any category $\mathbf{K}$ amounts to $P(X)$ in $\mathbf{K}^{op}$.

Since any category is the opposite of a certain category (namely, of its opposite), the following fact holds:

Fact 3.1.23 (Duality principle). *If W holds for all categories then* co-*W holds for all categories as well.* □

3.1.2.3 Product categories

Definition 3.1.24 (Product category). For any two categories **K1** and **K2**, the *product category* $\mathbf{K1} \times \mathbf{K2}$ is defined by:

Objects of $\mathbf{K1} \times \mathbf{K2}$: $|\mathbf{K1} \times \mathbf{K2}| = |\mathbf{K1}| \times |\mathbf{K2}|$ (the Cartesian product);
Morphisms of $\mathbf{K1} \times \mathbf{K2}$: for all $A, A' \in |\mathbf{K1}|$ and $B, B' \in |\mathbf{K2}|$,
$\mathbf{K1} \times \mathbf{K2}(\langle A, B\rangle, \langle A', B'\rangle) = \mathbf{K1}(A, A') \times \mathbf{K2}(B, B')$;
Composition in $\mathbf{K1} \times \mathbf{K2}$: for $f : A \to A'$ and $f' : A' \to A''$ in **K1**, $g : B \to B'$ and $g' : B' \to B''$ in **K2**, $\langle f, g\rangle ; \langle f', g'\rangle = \langle f ; f', g ; g'\rangle$. □

Exercise 3.1.25. Identify the category to which each semicolon in the above definition of composition in $\mathbf{K1} \times \mathbf{K2}$ refers. Then show that $\mathbf{K1} \times \mathbf{K2}$ is indeed a category. □

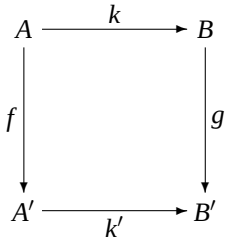
Exercise 3.1.26. Define $\mathbf{K}^n$, where $\mathbf{K}$ is a category and $n \geq 1$. What would you suggest for $n = 0$? □

3.1.2.4 Morphism categories

Definition 3.1.27 (Morphism category). For any category $\mathbf{K}$, the *category* $\mathbf{K}^{\to}$ *of* $\mathbf{K}$*-morphisms* is defined by:

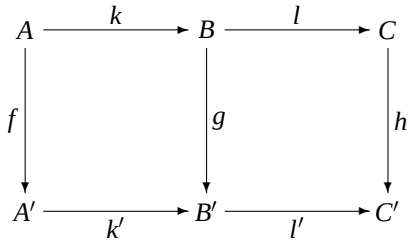
Objects of $\mathbf{K}^{\to}$: $\mathbf{K}$-morphisms;
Morphisms of $\mathbf{K}^{\to}$: a morphism in $\mathbf{K}^{\to}$ from $f : A \to A'$ (in $\mathbf{K}$) to $g : B \to B'$ (in $\mathbf{K}$) is a pair $\langle k, k'\rangle$ of $\mathbf{K}$-morphisms where $k : A \to B$ and $k' : A' \to B'$ such that $k ; g = f ; k'$;
Composition in $\mathbf{K}^{\to}$: $\langle k, k'\rangle ; \langle l, l'\rangle = \langle k ; l, k' ; l'\rangle$. □

The requirement in the definition of a morphism in $\mathbf{K}^{\to}$ may be more illustratively restated as the requirement that the following diagram commutes in the category $\mathbf{K}$:



For now, we will rely on an intuitive understanding of the concept of a diagram in a category; see Section 3.2.5 for a formal definition. We say that a diagram in a category *commutes* (or, *is commutative*) if for any two paths with the same source and target nodes, the composition of morphisms along each of the two paths yields the same result.

Drawing diagrams and *chasing* a diagram in order to prove that it is commutative is one of the standard and intuitively most appealing techniques used in category theory. For example, to justify Definition 3.1.27 above it is essential to show that the composition of two morphisms in $\mathbf{K}^{\rightarrow}$ as defined there yields a morphism in $\mathbf{K}^{\rightarrow}$. This may be done by *pasting together* two diagrams like the one above along a common edge, obtaining the following diagram:

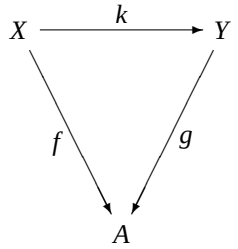


A simple argument may now be used to show that if the two simpler diagrams are commutative then the above diagram obtained by pasting them together along the edge labelled by g commutes as well:

$$f;(k';l') = (f;k');l' = (k;g);l' = k;(g;l') = k;(l;h) = (k;l);h$$

Definition 3.1.28 (Slice category). Let $\mathbf{K}$ be a category with $A \in |\mathbf{K}|$. The *category* $\mathbf{K}{\downarrow}A$ *of* $\mathbf{K}$*-objects over* A (or, the *slice of* $\mathbf{K}$ *over* A) is defined by:

Objects of $\mathbf{K}{\downarrow}A$: pairs $\langle X, f\rangle$ where $X \in |\mathbf{K}|$ and $f \in \mathbf{K}(X,A)$;
Morphisms of $\mathbf{K}{\downarrow}A$: a morphism from $\langle X, f\rangle$ to $\langle Y, g\rangle$ is a $\mathbf{K}$-morphism $k{:}X \to Y$ such that $k;g = f$:



Composition in $\mathbf{K}{\downarrow}A$: as in $\mathbf{K}$. □

Exercise 3.1.29. Show that $\mathbf{K}{\downarrow}A$ may be constructed as a subcategory of $\mathbf{K}^{\rightarrow}$. Is it full? □

Exercise 3.1.30. Define $\mathbf{K}{\uparrow}A$, the category of $\mathbf{K}$-objects *under* A. Compare $(\mathbf{K}{\downarrow}A)^{op}$, $\mathbf{K}^{op}{\downarrow}A$ and $(\mathbf{K}^{op}{\downarrow}A)^{op}$ with $\mathbf{K}{\uparrow}A$. □

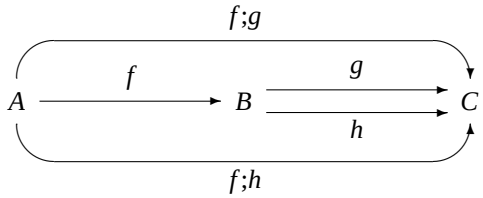
3.1.3 Category-theoretic definitions

In this section we will give a few simple examples of how certain special morphisms may be characterised in a style that is typical for category-theoretic definitions. As indicated in the introduction to this chapter, the idea is to abstract away from the "internal" properties of objects and morphisms, characterising them entirely in categorical language by referring only to arbitrary objects and morphisms of the category under consideration. Such definitions may be formulated for an arbitrary category, and then instantiated to a particular one when necessary. We will also indicate a few basic properties of the concepts we introduce that hold in any category.

Throughout this section, let **K** be an arbitrary but fixed category. Morphisms and objects we refer to below are those of **K**, unless explicitly qualified otherwise.

3.1.3.1 Epimorphisms and monomorphisms

Definition 3.1.31 (Epimorphism). A morphism $f: A \to B$ is an *epimorphism* (or *is epi*) if for all $g: B \to C$ and $h: B \to C$, $f;g = f;h$ implies $g = h$.



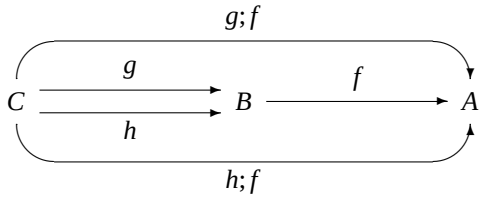
□

Example 3.1.32. In **Set**, f is epi iff f is surjective. □

There are "natural" categories in which epimorphisms need not be surjective. For example:

Exercise 3.1.33. Recall the category **CPO** of complete partial orders and continuous functions introduced in Example 3.1.9. Give an example of a continuous function that is an epimorphism in **CPO** even though it is not surjective. Try to characterise epimorphisms in this category. □

Definition 3.1.34 (Monomorphism). A morphism $f: B \to A$ is a *monomorphism* (or *is mono*) if for all $g: C \to B$ and $h: C \to B$, $g;f = h;f$ implies $g = h$.



□

Example 3.1.35. In **Set**, f is mono iff f is injective. □

Note that mono means the same as co-epi, i.e., f is mono in $\mathbf{K}$ iff f is epi in $\mathbf{K}^{op}$.

Fact 3.1.36.

1. *If $f:A \to B$ and $g:B \to C$ are mono then $f;g:A \to C$ is mono.*
2. *For any $f:A \to B$ and $g:B \to C$, if $f;g:A \to C$ is mono then f is mono.*

Proof. The proof is rather straightforward, and significantly more complex proofs will be omitted in the rest of this chapter. We present it here explicitly only as a simple example of the style of argument, very common in category-theoretic proofs, exploiting the most basic properties of composition in an arbitrary category.

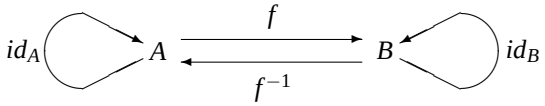
1. According to Definition 3.1.34, we have to show that for any $h,h':D \to A$ if $h;(f;g) = h';(f;g)$ then $h = h'$. So, suppose $h;(f;g) = h';(f;g)$. Then, since composition is associative, $(h;f);g = (h';f);g$. Consequently, since g is mono, by Definition 3.1.34, $h;f = h';f$. Thus, using the fact that f is mono, we can indeed derive $h = h'$.
2. Similarly as in the previous case: suppose that for some $h,h':D \to A$, $h;f = h';f$. Then also $(h;f);g = (h';f);g$, and so $h;(f;g) = h';(f;g)$. Now, since $f;g$ is mono, it follows directly from the definition that indeed $h = h'$.

□

Exercise 3.1.37. Dualise both parts of Fact 3.1.36. Formulate the dual proofs and check that they are indeed sound. □

3.1.3.2 Isomorphic objects

Definition 3.1.38 (Isomorphism). A morphism $f:A \to B$ is an *isomorphism* (or *iso*) if there is a morphism $f^{-1}:B \to A$ such that $f;f^{-1} = id_A$ and $f^{-1};f = id_B$. The morphism $f^{-1}:B \to A$ is called the *inverse* of f, and the objects A and B are called *isomorphic*. We write $f:A \cong B$ or just $A \cong B$.



□

Exercise 3.1.39. Show that the inverse of a morphism, if it exists, is unique. □

Note that iso means the same as co-iso, that is, isomorphism is a *self-dual* concept.

Exercise 3.1.40. Check that if $f:A \to B$ and $g:B \to C$ are iso then $f;g:A \to C$ is iso as well. □

In **Set**, a morphism is iso iff it is both epi and mono. However, this property does not carry over to an arbitrary category:

Exercise 3.1.41. Show that if f is iso then f is both epi and mono. The converse is not true in general; give a counterexample. □

Exercise 3.1.42. We say that a morphism $f:A \to B$ is a *retraction* if there is a morphism $g:B \to A$ such that $g;f = id_B$. Dually, a morphism $f:A \to B$ is a *coretraction* if there is a morphism $g:B \to A$ such that $f;g = id_A$. Show that:

1. A morphism is iso iff it is both a retraction and a coretraction.
2. Every retraction is epi.
3. A morphism is iso iff it is an epi coretraction.

Dualise the above facts. □

It is easy to see that any two isomorphic objects have the same "categorical properties". Intuitively, such objects have abstractly the same structure and so are indistinguishable within the given category (which does not mean that isomorphic objects cannot have different "non-categorical" properties, cf. Example 1.3.12). Indeed, an isomorphism and its inverse determine one-to-one mappings between morphisms going into and coming out of isomorphic objects. Hence, categorical definitions of objects define them only "up to isomorphism". The following section provides typical examples of this phenomenon.

3.2 Limits and colimits

In this section we show how certain special objects in an arbitrary category together with their "characteristic" morphisms may be defined in purely categorical terms by so-called *universal properties*; we hope that the reader will recognise the pattern in the example definitions below. Sections 3.2.1–3.2.4 present some typical instances of this, introducing the most commonly used cases of the general *limit construction* and its dual, which are then presented in their full generality in Section 3.2.5. In most of the cases in this section we will explicitly spell out the duals of the concepts introduced, since many of them have interesting instances in some common categories (and are traditionally given independent names).

Throughout this section, let **K** be an arbitrary but fixed category. Morphisms and objects we refer to are those of **K**, unless explicitly qualified otherwise.

3.2.1 Initial and terminal objects

Definition 3.2.1 (Initial object). An object $I \in |\mathbf{K}|$ is *initial in* **K** if for each $A \in |\mathbf{K}|$ there is exactly one morphism from I to A. □

Example 3.2.2. The empty set $\varnothing$ is initial in **Set**. The algebra T_Σ of ground Σ-terms is initial in $\mathbf{Alg}(\Sigma)$, for any signature $\Sigma \in |\mathbf{AlgSig}|$.

Recall the definition of an initial model of an equational specification (Definition 2.5.13). For any signature Σ and a set Φ of Σ-equations, the initial model of $\langle \Sigma, \Phi \rangle$ (which exists by Theorem 2.5.14) is an initial object in the category $\mathbf{Mod}(\Sigma, \Phi)$ (as defined in Example 3.1.19). □

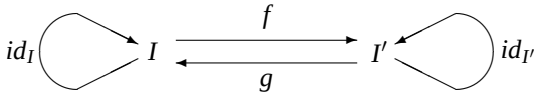
Exercise 3.2.3. What is an initial object in **AlgSig**? Look for initial objects in other categories. □

Fact 3.2.4.

1. Any two initial objects in $\mathbf{K}$ *are isomorphic.*
2. If I *is initial in* $\mathbf{K}$ *and* I' *is isomorphic to* I *then* I' *is initial in* $\mathbf{K}$ *as well.*

Proof. The proof is rather straightforward. We present it here explicitly only as a simple example of the style of argument, very common in category-theoretic proofs, which exploits universality (a special case of which is the property used in the definition of an initial object). The requirement that there *exists* a morphism satisfying a certain property is used to construct some diagrams, and then the *uniqueness* of this morphism is used to show that the diagrams constructed commute.

1. Suppose that $I, I' \in |\mathbf{K}|$ are two initial objects in $\mathbf{K}$. Then, by the initiality of I, there exists a morphism $f : I \to I'$. Similarly, by the initiality of I', there exists a morphism $g : I' \to I$. Thus, we have constructed the following diagram:



   Now, by the initiality of I, there is a *unique* morphism from I to I, and so $id_I = f;g$. Similarly, $id_{I'} = g;f$. Thus f is an isomorphism (with inverse g) and I and I' are indeed isomorphic.
2. Suppose that $I \in |\mathbf{K}|$ is initial in $\mathbf{K}$, and let $i : I \to I'$ be an isomorphism with inverse $i^{-1} : I' \to I$. Consider an arbitrary object $A \in |\mathbf{K}|$. By the "existence part" of the initiality property of I, we know that there exists a morphism $f : I \to A$. Hence, there exists a morphism from I' to A as well, namely $i^{-1};f : I' \to A$. Then, let $f' : I' \to A$ be an arbitrary morphism from I' to A. By the "uniqueness part" of the initiality property of I, $f = i;f'$, and so $i^{-1};f = i^{-1};(i;f') = (i^{-1};i);f' = id_{I'};f' = f'$. This shows that $i^{-1};f$ is the only morphism from I' to A, and so that I' is indeed initial in $\mathbf{K}$.

□

The last fact indicates that the initiality property identifies an object up to isomorphism. As argued in Section 3.1.3.2, in category theory this is the most exact characterisation of an object we may expect. In the following we will speak of "the" initial object meaning an initial object identified up to isomorphism. We adopt the same convention in the many similar cases introduced in the sequel.

3.2.1.1 Dually:

Definition 3.2.5 (Terminal object). An object $1 \in |\mathbf{K}|$ is *terminal in* $\mathbf{K}$ if for each $A \in |\mathbf{K}|$ there is exactly one morphism from A to 1. □

Note that terminal means the same as co-initial.

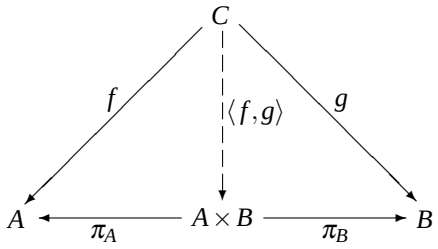
Exercise 3.2.6. Are there any terminal objects in **Set**, $\mathbf{Alg}(\Sigma)$ or **AlgSig**? What about terminal objects in $\mathbf{AlgSig}^{der}$?

Recall the definition of a terminal (final) model of an equational specification (Definition 2.7.12). Restate it using the notion of a terminal object as defined above. □

Exercise 3.2.7. Dualise Fact 3.2.4. □

3.2.2 Products and coproducts

Definition 3.2.8 (Product). A *product* of two objects $A, B \in |\mathbf{K}|$ is an object $A \times B \in |\mathbf{K}|$ together with a pair of morphisms $\pi_A : A \times B \to A$ and $\pi_B : A \times B \to B$ such that for any object $C \in |\mathbf{K}|$ and pair of morphisms $f : C \to A$ and $g : C \to B$ there is exactly one morphism $\langle f, g \rangle : C \to A \times B$ such that the following diagram commutes:



□

Example 3.2.9. In **Set**, the Cartesian product of A and B is a product $A \times B$, where π_A, π_B are the projection functions. For any signature Σ, products in $\mathbf{Alg}(\Sigma)$ are defined analogously (cf. Definition 1.2.9). □

Exercise 3.2.10. What is the product of two objects in a preorder category? □

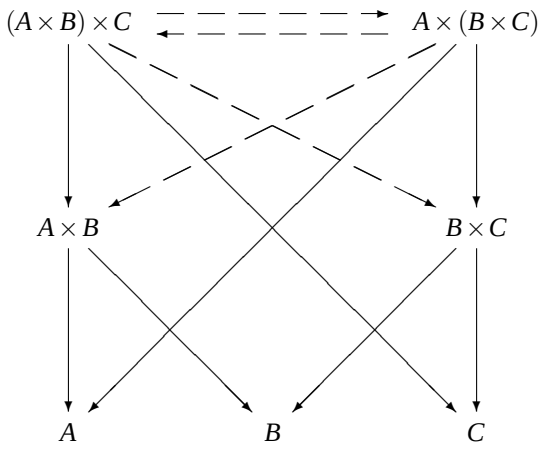
Exercise 3.2.11. Show that any two products of $A, B \in |\mathbf{K}|$ are isomorphic. □

Exercise 3.2.12. Suppose that $A, B \in |\mathbf{K}|$ have a product. Given $f : C \to A$ and $g : C \to B$, and hence $\langle f, g \rangle : C \to A \times B$, show that for any $h : D \to C$, $h ; \langle f, g \rangle = \langle h ; f, h ; g \rangle$. □

Exercise 3.2.13. Prove that:

1. $A \times B \cong B \times A$ for any $A, B \in |\mathbf{K}|$.

2. $(A \times B) \times C \cong A \times (B \times C)$ for any $A, B, C \in |\mathbf{K}|$. HINT: The following diagram might be helpful:

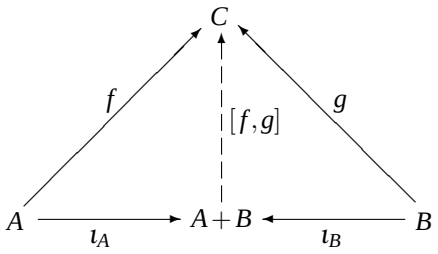


□

Exercise 3.2.14. Define the product of an arbitrary family of **K**-objects. What is the product of the empty family? □

3.2.2.1 Dually:

Definition 3.2.15 (Coproduct). A *coproduct* of two objects $A, B \in |\mathbf{K}|$ is an object $A+B \in |\mathbf{K}|$ together with a pair of morphisms $\iota_A : A \to A+B$ and $\iota_B : B \to A+B$ such that for any object $C \in |\mathbf{K}|$ and pair of morphisms $f : A \to C$ and $g : B \to C$ there is exactly one morphism $[f,g] : A+B \to C$ such that the following diagram commutes:



□

Example 3.2.16. In **Set**, the disjoint union of sets A and B is their coproduct $A+B$, where ι_A, ι_B are the injections. Similarly, in **AlgSig**, the (componentwise) disjoint union of algebraic signatures Σ and Σ' is their coproduct $\Sigma + \Sigma'$, where ι_A, ι_B are the obvious injections. □

Note that coproduct means the same as co-product.

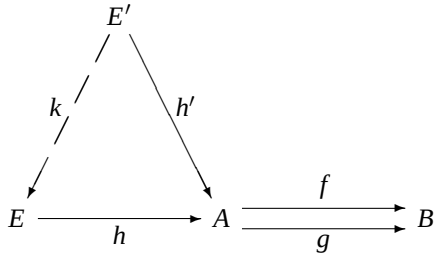
Exercise 3.2.17. Dualise the exercises for products. □

Exercise 3.2.18. For any algebraic signature $\Sigma = \langle S, \Omega \rangle$ and two S-sorted sets X and Y, show that their disjoint union $X \uplus Y$ is the coproduct of X and Y in the category $\mathbf{T}_\Sigma$ of substitutions over Σ (recall Example 3.1.13), where the coproduct injections are the identity substitutions (of the corresponding variables from $X \uplus Y$ for variables in X and in Y, respectively). Generalise this to the category $\mathbf{T}_\Sigma/\Phi$ of substitutions over Σ modulo a set Φ of Σ-equations (cf. Exercise 3.1.14). Finally, characterise coproducts in the category $\mathbf{T}_{\Sigma,\Phi}$, the algebraic theory over Σ generated by Φ (Exercise 3.1.15). □

3.2.3 Equalisers and coequalisers

We have defined above products and coproducts for arbitrary pairs of objects in a category. In this section we deal with constructions for pairs of morphisms constrained to be *parallel*, i.e., pairs of morphisms that have the same source and the same target.

Definition 3.2.19 (Equaliser). An *equaliser* of two parallel morphisms $f: A \to B$ and $g: A \to B$ is an object $E \in |\mathbf{K}|$ together with a morphism $h: E \to A$ such that $h; f = h; g$, and such that for any object $E' \in |\mathbf{K}|$ and morphism $h': E' \to A$ satisfying $h'; f = h'; g$ there is exactly one morphism $k: E' \to E$ such that $k; h = h'$:



□

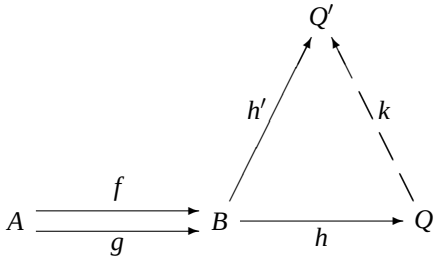
Exercise 3.2.20. Show that an equaliser of $f: A \to B$ and $g: A \to B$ is unique up to isomorphism. □

Exercise 3.2.21. Show that every equaliser (to be more precise: its morphism part) is mono, and every epi equaliser is iso. □

Exercise 3.2.22. Construct equalisers of pairs of parallel morphisms in **Set**. Then, for any signature Σ, construct equalisers of pairs of parallel morphisms in $\mathbf{Alg}(\Sigma)$. HINT: For any two functions $f, g: A \to B$ consider the set $\{a \in A \mid f(a) = g(a)\} \subseteq A$. □

3.2.3.1 Dually:

Definition 3.2.23 (Coequaliser). The dual notion to equaliser is *coequaliser*. The diagram now looks as follows:



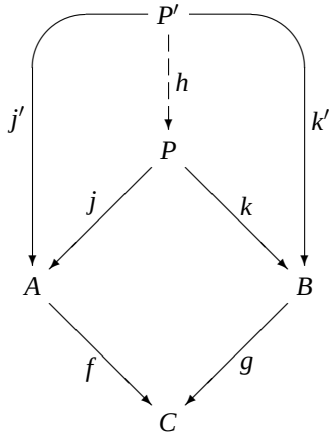
Exercise. Formulate explicitly the definition of a coequaliser. Then dualise the exercises for equalisers. □

Exercise 3.2.24. What is the coequaliser of two morphisms in **Set**? What is the coequaliser of two morphisms in **AlgSig**? What is the coequaliser of two morphisms in $\mathbf{Alg}(\Sigma)$? HINT: Given two functions $f,g:A\to B$ consider the quotient of B by the least equivalence relation $\equiv$ on B such that for all $a\in A$, $f(a)\equiv g(a)$. □

Exercise 3.2.25. What is the coequaliser of two morphisms in the category of substitutions $\mathbf{T}_\Sigma$? □

3.2.4 Pullbacks and pushouts

Definition 3.2.26 (Pullback). A *pullback* of two morphisms $f:A\to C$ and $g:B\to C$ having the same codomain is an object $P\in|\mathbf{K}|$ together with a pair of morphisms $j:P\to A$ and $k:P\to B$ such that $j;f=k;g$, and such that for any object $P'\in|\mathbf{K}|$ and pair of morphisms $j':P'\to A$ and $k':P'\to B$ satisfying $j';f=k';g$ there is exactly one morphism $h:P'\to P$ such that the following diagram commutes:



□

Exercise 3.2.27. Show that a pullback of $f:A\to C$ and $g:B\to C$ is unique up to isomorphism. □

Exercise 3.2.28. Show that if **K** has products (of all pairs of objects) and equalisers (of all pairs of parallel morphisms) than it has pullbacks as well (i.e., all pairs of morphisms with common target have pullbacks in **K**).

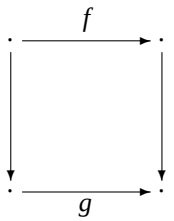
HINT: To construct a pullback of $f:A \to C$ and $g:B \to C$, first construct the product $A \times B$ with projections $\pi_A:A \times B \to A$ and $\pi_B:A \times B \to B$ and then the equaliser $h:P \to A \times B$ of $\pi_A;f:A \times B \to C$ and $\pi_B;g:A \times B \to C$. □

Exercise 3.2.29. Construct the pullback of two morphisms in **Set**, then in **Alg**(Σ), and in **AlgSig**. □

Exercise 3.2.30. Prove that if **K** has a terminal object and all pullbacks (i.e., any pair of **K**-morphisms with common target has a pullback in **K**) then:

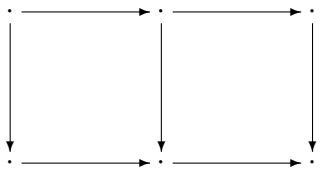
1. **K** has all (binary) products.
2. **K** has all equalisers. HINT: Get the equaliser of $f,g:A \to B$ from the pullback of $\langle id_A, f\rangle, \langle id_A, g\rangle:A \to A \times B$. □

Exercise 3.2.31. Prove that pullbacks translate monomorphisms to monomorphisms: if



is a pullback square and g is mono, then f is mono as well. □

Exercise 3.2.32. Consider the following diagram:

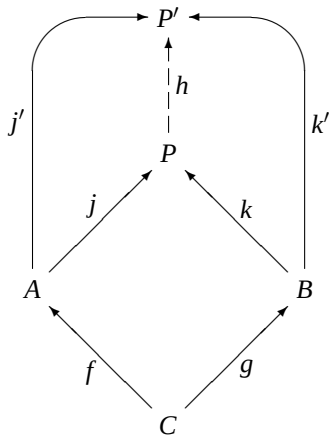


Prove that:

1. If the two squares are pullbacks then the outer rectangle is a pullback.
2. If the diagram commutes and the outer rectangle and right-hand square are both pullbacks then so is the left-hand square. □

3.2.4.1 Dually:

Definition 3.2.33 (Pushout). The dual notion to pullback is *pushout*. The diagram now looks as follows:



Exercise. Spell out the definition of a pushout explicitly. Then dualise the exercises for pullbacks. □

Pushouts provide a basic tool for "putting together" structures of different kinds. Given two objects A and B, a pair of morphisms $f{:}C \to A$ and $g{:}C \to B$ indicates a common source from which some "parts" of A and B come. The pushout of f and g puts together A and B while identifying the parts coming from the common source as indicated by f and g, but keeping the new parts disjoint (cf. the dual of Exercise 3.2.28).

Example 3.2.34. Working in **Set**, consider:

$$\begin{array}{ll} A = \{1,2,3\} & \\ B = \{3,4,5\} & \\ C = \{\clubsuit\} & \\ f = \{\clubsuit \mapsto 2\} & :C \to A \\ g = \{\clubsuit \mapsto 4\} & :C \to B \end{array}$$

Then the pushout object P is (up to isomorphism) given as follows:

$$\begin{array}{ll} P = \{1', \{2'{=}4''\}, 3', 3'', 5''\} & \\ j = \{1 \mapsto 1', 2 \mapsto \{2'{=}4''\}, 3 \mapsto 3'\} & :A \to P \\ k = \{3 \mapsto 3'', 4 \mapsto \{2'{=}4''\}, 5 \mapsto 5''\} & :B \to P \end{array}$$ □

Example 3.2.35. The general comments above about the use of pushouts for putting together objects in categories apply in particular when one wants to combine algebraic signatures, as we will frequently do throughout the rest of the book. As a very simple example of a pushout in the category **AlgSig** of algebraic signatures, consider the signature $\Sigma\mathrm{NAT}$ of natural numbers defined in Exercise 2.5.4. Then, let $\Sigma\mathrm{NAT}_{fib}$ be its extension by a new operation name $fib{:}\,nat \to nat$ and $\Sigma\mathrm{NAT}_{mult}$ its extension by another operation name $mult{:}\,nat \times nat \to nat$. We then have two signature inclusions:

$$\Sigma\mathrm{NAT}_{fib} \hookleftarrow \Sigma\mathrm{NAT} \hookrightarrow \Sigma\mathrm{NAT}_{mult}$$

Their pushout in **AlgSig** yields a signature $\Sigma\mathrm{NAT}_{fib,mult}$ which (up to isomorphism) consists of the shared signature $\Sigma\mathrm{NAT}$ (once, no repetitions!) together with each of the operations added by the two extensions.

This is deceptively simple though, involving only single-sorted signature inclusions that introduce different operation names.

Exercise. Give examples of pushouts in **AlgSig** with signatures involving more than one sort, operation names that coincide, and signature morphisms that are not injective on sorts and/or on operation names. □

3.2.5 The general situation

The definitions introduced in the previous subsections followed a common, more general pattern. As an example, let's have another look at the definition of a pullback (Definition 3.2.26; the notation below refers to the diagram there). Given a diagram in the category at hand (the two morphisms f and g of which we construct the pullback), we consider an object P in this category together with morphisms going from this object to the nodes of the diagram (j, k and an anonymous $c\colon P \to C$) such that all the resulting paths starting from P commute ($j;f = c = k;g$ — hence c may remain anonymous). Moreover, from among all such objects we choose the one that is in a sense "closest" to the diagram: for any object P' with morphisms from it to the diagram nodes (j', k' and an anonymous c') satisfying the required commutativity property ($j';f = c' = k';g$), P' may be uniquely projected onto the chosen object P (via a morphism h) so that all the resulting paths starting from P' commute ($h;j = j'$ and $h;k = k'$, which also implies $h;c = c'$). This is usually referred to as the *universal property* of pullbacks and, more generally, of arbitrary *limits* as defined below. The (dual) universal property of pushouts and, more generally, of arbitrary *colimits* as defined below, may be described by looking at objects with morphisms going from the nodes of a diagram into them. We will formalise this in the rest of this section.

Definition 3.2.36 (Graph). Let Σ_G be the following signature:

sorts *node*, *edge*
ops $source\colon edge \to node$
$target\colon edge \to node$

A Σ_G-algebra is called a *graph*. (Note that these graphs may have multiple edges between any two nodes; such graphs are sometimes called *multigraphs*.) The category **Graph** of graphs is $\mathbf{Alg}(\Sigma_G)$. Given a graph G, we write $e\colon n \to m$ as an abbreviation for $n, m \in |G|_{node}$, $e \in |G|_{edge}$, $source_G(e) = n$ and $target_G(e) = m$. □

Exercise 3.2.37. Construct an initial object, coproducts, coequalisers and pushouts in **Graph**. □

Exercise 3.2.38. Define formally the category $\mathbf{Path}(G)$ of paths in a graph G, where:

Objects of $\mathbf{Path}(G)$: $|G|_{node}$;
Morphisms of $\mathbf{Path}(G)$: paths in G, i.e., finite sequences $e_1 \dots e_n$ of elements of $|G|_{edge}$ such that $source_G(e_{i+1}) = target_G(e_i)$ for $i < n$. Notice that we have to allow for $n = 0$. □

A diagram in **K** is a graph having nodes labelled with **K**-objects and edges labelled with **K**-morphisms with the appropriate source and target. Formally:

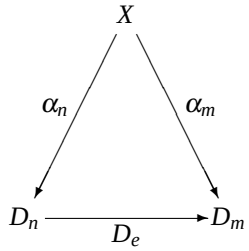
Definition 3.2.39 (Diagram). A *diagram* D in **K** consists of:

- a graph $G(D)$;
- for each node $n \in |G(D)|_{node}$, an object $D_n \in |\mathbf{K}|$; and
- for each edge $e: n \to m$ in $G(D)$, a morphism $D_e: D_n \to D_m$.

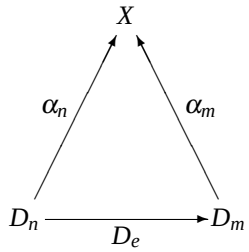
A diagram D is *connected* if its graph $G(D)$ is connected (that is, any two nodes in $G(D)$ are linked by a sequence of edges disregarding their direction, or fully formally: if the total relation on the set of nodes of $G(D)$ is the only equivalence between the nodes that links all nodes having an edge between them). □

Exercise 3.2.40. Show how every small category **K** gives rise to a graph $G(\mathbf{K})$ and a diagram $D(\mathbf{K})$. □

Definition 3.2.41 (Cone and cocone). A *cone* α *over a diagram* D *in* **K** is a **K**-object X together with a family of **K**-morphisms $\langle \alpha_n: X \to D_n \rangle_{n \in |G(D)|_{node}}$ such that for every edge $e: n \to m$ in the graph $G(D)$ the following diagram commutes:



Dually: A *cocone* α *over a diagram* D *in* **K** is a **K**-object X together with a family of **K**-morphisms $\langle \alpha_n: D_n \to X \rangle_{n \in |G(D)|_{node}}$ such that for every edge $e: n \to m$ in the graph $G(D)$ the following diagram commutes:

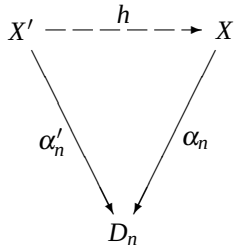


□

In the following we will write cones simply as families $\langle \alpha_n : X \to D_n \rangle_{n \in |G(D)|_{node}}$, omitting any explicit mention of the apex X, and similarly for cocones. The notation is not quite justified only in the case when the diagram (and hence the family) is empty; this will not lead to any misunderstanding.

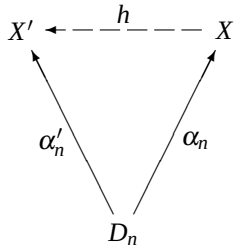
Let D be a diagram in $\mathbf{K}$ with $|G(D)|_{node} = N$ and $|G(D)|_{edge} = E$.

Definition 3.2.42 (Limit and colimit). A *limit of D in* $\mathbf{K}$ is a cone $\langle \alpha_n : X \to D_n \rangle_{n \in N}$ such that for any cone $\langle \alpha'_n : X' \to D_n \rangle_{n \in N}$ there is exactly one morphism $h : X' \to X$ such that for every $n \in N$ the following diagram commutes:



If $\langle \alpha_n : X \to D_n \rangle_{n \in N}$ is a limit of D, we will refer to X as the *limit object* of D (or sometimes just the *limit* of D), and to the morphisms α_n, $n \in N$, as the limit *projections*.

Dually: A *colimit of D in* $\mathbf{K}$ is a cocone $\langle \alpha_n : D_n \to X \rangle_{n \in N}$ such that for any cocone $\langle \alpha'_n : D_n \to X' \rangle_{n \in N}$ there is exactly one morphism $h : X \to X'$ such that for every $n \in N$ the following diagram commutes:



If $\langle \alpha_n : D_n \to X \rangle_{n \in N}$ is a colimit of D, we will refer to X as the *colimit object* of D (or sometimes just the *colimit* of D), and to the morphisms α_n, $n \in N$, as the colimit *injections*. □

Definition 3.2.43 (Completeness and cocompleteness). A category $\mathbf{K}$ is *(finitely) complete* if every (finite) diagram in $\mathbf{K}$ has a limit. Dually, $\mathbf{K}$ is *(finitely) cocomplete* if every (finite) diagram in $\mathbf{K}$ has a colimit. □

Exercise 3.2.44. Define formally the category $\mathbf{Cone}(D)$ of cones over a diagram D, where:

Objects of $\mathbf{Cone}(D)$: cones over D;
Morphisms of $\mathbf{Cone}(D)$: a morphism from $\alpha = \langle \alpha_n : X \to D_n \rangle_{n \in N}$ to $\alpha' = \langle \alpha'_n : X' \to D_n \rangle_{n \in N}$ is a $\mathbf{K}$-morphism $h : X \to X'$ such that $\alpha_n = h ; \alpha'_n$ for $n \in N$.

Prove that the limit of D is a terminal object in $\mathbf{Cone}(D)$. Note that this implies that a limit of any diagram is unique up to isomorphism.

Present the category of objects over an object (cf. Definition 3.1.28) as the category of cones over a certain diagram. □

Exercise 3.2.45. Show that products, terminal objects, equalisers and pullbacks in $\mathbf{K}$ are limits of simple diagrams in $\mathbf{K}$. □

Exercise 3.2.46. Construct in **Set** a limit of the diagram

$$A_0 \xleftarrow{f_0} A_1 \xleftarrow{f_1} A_2 \xleftarrow{f_2} A_3 \xleftarrow{f_3} \cdots$$ □

Exercise 3.2.47. Show that limiting cones are *jointly mono*: if $\langle \alpha_n : X \to D_n \rangle_{n \in |G(D)|_{node}}$ is a limit of D, then $f = g$ whenever for all $n \in |G(D)|_{node}$, $f;\alpha_n = g;\alpha_n$. □

Exercise 3.2.48. Show that if $\mathbf{K}$ has a terminal object, binary products and all equalisers then it is finitely complete. HINT: Given a finite diagram in $\mathbf{K}$, first build the product of all its objects, and then gradually turn it into a limit by "equalising" the triangles formed by product projections and morphisms in the diagram.

Use Exercise 3.2.30 to conclude that if $\mathbf{K}$ has a terminal object and all pullbacks then it is finitely complete. □

Exercise 3.2.49. Show that if $\mathbf{K}$ has products of arbitrary families of objects and all equalisers then it is complete. HINT: Proceed as in Exercise 3.2.48, but notice that all the triangles involved may be "equalised" simultaneously in one step, cf. [Mac71], Theorem V.2.1. □

Exercise 3.2.50. A *wide pullback* is the limit of a non-empty family of morphisms with a common target. Show that if a category has a terminal object and all wide pullbacks then it has products of arbitrary families of objects, and then conclude that it is complete. HINT: Generalise Exercise 3.2.30 and use Exercise 3.2.49. □

Exercise 3.2.51. Recall that for any category $\mathbf{K}$ and object $A \in |\mathbf{K}|$, $\mathbf{K}{\downarrow}A$ is the slice category of objects over A (Definition 3.1.28).

Notice that $\mathbf{K}{\downarrow}A$ has a terminal object. Then show that binary products in $\mathbf{K}{\downarrow}A$ are essentially given by the pullbacks in $\mathbf{K}$ (of morphisms to A) and similarly, arbitrary non-empty products in $\mathbf{K}{\downarrow}A$ are essentially given by wide pullbacks in $\mathbf{K}$. Check also that any (wide) pullback in $\mathbf{K}{\downarrow}A$ is given by the corresponding (wide) pullback in $\mathbf{K}$ (no morphisms to A added).

Conclude that $\mathbf{K}{\downarrow}A$ is finitely complete if $\mathbf{K}$ has all pullbacks, and $\mathbf{K}{\downarrow}A$ is complete if $\mathbf{K}$ has all wide pullbacks. □

Exercise 3.2.52. Dualise the above exercises. □

Exercise 3.2.53. Show that:

1. **Set** is complete and cocomplete.

2. **FinSet** is finitely complete and finitely cocomplete, but is neither complete nor cocomplete.
3. **Alg**(Σ) is complete for any signature Σ. (It is also cocomplete, but the proof is harder — give it a try!)
4. **AlgSig** is cocomplete. (Is it complete?)

HINT: Use Exercise 3.2.49 and its dual, and the standard constructions of (co)products and (co)equalisers in these categories hinted at in Examples 3.2.9, 3.2.16 and Exercises 3.2.22, 3.2.24. Check that, given a diagram D with nodes N and edges E in **Set**, its limit is (up to isomorphism) the set of families $\langle d_n \rangle_{n \in N}$ that are compatible with D in the sense that $d_n \in D_n$ for each $n \in N$ and $d_m = D_e(d_n)$ for each edge $e: n \to m$, with the obvious projections. Check that its colimit is (up to isomorphism) the quotient of the disjoint union $\biguplus_{n \in N} D_n$ by the least equivalence relation that is generated by all pairs $\langle d_n, D_e(d_n) \rangle$ for $e: n \to m$ in E and $d_n \in D_n$. □

Exercise 3.2.54. Show that $\mathbf{AlgSig}^{der}$ is not finitely cocomplete. (HINT: Consider a morphism mapping a binary operation to the projection on the first argument and another morphism mapping the same operation to the projection on the second argument. Can such a pair of morphisms have a coequaliser?) □

Exercise 3.2.55. When is a preorder category (finitely) complete and cocomplete? □

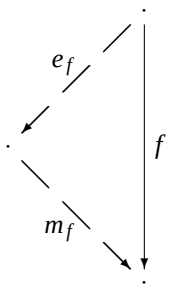
3.3 Factorisation systems

In this section we will interrupt our presentation of the basic concepts of category theory and try to illustrate how they can be used to formulate some well-known ideas at a level of generality and abstraction that ensures their applicability in many specific contexts.

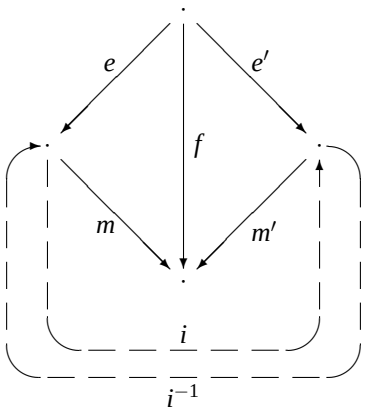
The concept on which we concentrate here is that of *reachability* (cf. Section 1.2). Recall that the original definition of a reachable algebra used the notion of a subalgebra (cf. Definition 1.2.7). Keeping in mind that in the categorical framework we deal with objects identified up to isomorphism, we slightly generalise the standard formulation and, for any signature $\Sigma \in |\mathbf{AlgSig}|$, say that a Σ-algebra B is a subalgebra of A if there exists an *injective* Σ-homomorphism from B to A. A dual notion is that of a *quotient*: a Σ-algebra B is a quotient of a Σ-algebra A if there exists a *surjective* Σ-homomorphism from A to B. Now, a Σ-algebra A is *reachable* if it has no proper subalgebra (i.e., every subalgebra of A is isomorphic to A), or equivalently, if it is a quotient of the algebra T_Σ of ground Σ-terms (cf. Exercise 1.4.14). In this formulation, the above definitions may be used to introduce a notion of reachability in an arbitrary category. However, we need an appropriate generalisation of the concept of injective and surjective homomorphisms. A first attempt might be to use arbitrary epimorphisms and monomorphisms for this purpose, but it soon turns out that these concepts are not "fine enough" to ensure the properties we are after. An appropriate refinement of these is given if the category is equipped with a *factorisation system*.

Definition 3.3.1 (Factorisation system). Let **K** be an arbitrary category. A *factorisation system* for **K** is a pair $\langle \mathbf{E}, \mathbf{M} \rangle$, where:

- **E** is a collection of epimorphisms in **K** and **M** is a collection of monomorphisms in **K**;
- each of **E** and **M** is closed under composition and contains all isomorphisms in **K**;
- every morphism in **K** has an $\langle \mathbf{E}, \mathbf{M} \rangle$-factorisation: for each $f \in \mathbf{K}$, $f = e_f;m_f$ for some $e_f \in \mathbf{E}$ and $m_f \in \mathbf{K}$;



- $\langle \mathbf{E}, \mathbf{M} \rangle$-factorisations are unique up to isomorphism: for any $e, e' \in \mathbf{E}$ and $m, m' \in \mathbf{M}$, if $e;m = e';m'$ then there exists an isomorphism i such that $e;i = e'$ and $i;m' = m$.



□

Example 3.3.2. Set has a factorisation system $\langle \mathbf{E}, \mathbf{M} \rangle$, where **E** is the collection of all surjective functions and **M** is the collection of all injective functions. □

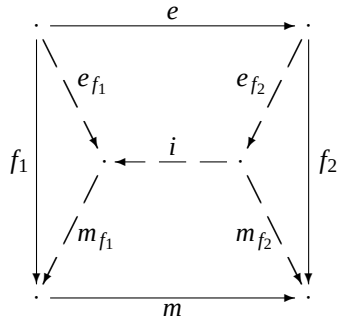
Example 3.3.3. For any signature Σ, $\mathbf{Alg}(\Sigma)$ has a factorisation system[4] $\langle \mathbf{TE}_\Sigma, \mathbf{TM}_\Sigma \rangle$, where $\mathbf{TE}_\Sigma$ is the collection of all surjective Σ-homomorphisms and $\mathbf{TM}_\Sigma$ is the collection of all injective Σ-homomorphisms; see Exercise 1.3.23. □

Consider an arbitrary category **K** equipped with a factorisation system $\langle \mathbf{E}, \mathbf{M} \rangle$.

[4] "**T**" in $\mathbf{TE}_\Sigma$ and $\mathbf{TM}_\Sigma$ indicates that we are dealing with ordinary *total* algebras here, as opposed to partial and continuous algebras with the factorisation systems discussed below.

Lemma 3.3.4 (Diagonal fill-in lemma). *For any morphisms f_1, f_2, e, m in* **K**, *where $e \in \mathbf{E}$ and $m \in \mathbf{M}$, if $f_1;m = e;f_2$ then there exists a unique morphism g such that $e;g = f_1$ and $g;m = f_2$.*

Proof sketch. The required "diagonal" is given by $g = e_{f_2};i;m_{f1}$, as illustrated by the diagram below; its uniqueness follows easily since e is an epimorphism.



□

Exercise 3.3.5. Show that if $e \in \mathbf{E}$ and $e;f \in \mathbf{M}$ for some morphism $f \in \mathbf{K}$, then e is an isomorphism. Dually, if $m \in \mathbf{M}$ and $f;m \in \mathbf{E}$ for some morphism $f \in \mathbf{K}$, then m is an isomorphism. □

Definition 3.3.6 (Subobject and quotient). Let $A \in |\mathbf{K}|$. A *subobject* of A is an object $B \in |\mathbf{K}|$ together with a morphism $m\colon B \to A$ such that $m \in \mathbf{M}$. A *quotient* of A is an object $B \in |\mathbf{K}|$ together with a morphism $e\colon A \to B$ such that $e \in \mathbf{E}$. □

Definition 3.3.7 (Reachable object). An object $A \in |\mathbf{K}|$ is *reachable* if it has no proper subobject, i.e., if every morphism $m \in \mathbf{M}$ with target A is an isomorphism. □

The category $\mathbf{Alg}(\Sigma)$ of Σ-algebras and the notion of a reachable algebra provide an instance of the general concept of reachability introduced in the above definition. The following theorem gives more general versions of well-known facts often laboriously proved in the standard algebraic framework.

Theorem 3.3.8. *Assume that* **K** *has an initial object Λ. Then:*

1. *An object $A \in |\mathbf{K}|$ is reachable iff it is a quotient of the initial object Λ.*
2. *Every object in $|\mathbf{K}|$ has a reachable subobject which is unique up to isomorphism.*
3. *If $A \in |\mathbf{K}|$ is reachable then for every $B \in |\mathbf{K}|$ there exists at most one morphism from A to B.*
4. *If $A \in |\mathbf{K}|$ is reachable and $f \in \mathbf{K}$ is a morphism with target A then $f \in \mathbf{E}$.* □

Exercise 3.3.9. Prove the theorem and identify the familiar facts about reachable algebras generalised here. □

One of the main results of Chapter 2, Theorem 2.5.14, states that any equational specification has an initial model. This is just a special case of a more general result which we formulate and prove for an arbitrary category with "reachability structure" satisfying an additional, technical property that any object has up to isomorphism only a *set* of quotients.

Definition 3.3.10 (Co-well-powered category). $\mathbf{K}$ is $\mathbf{E}$-*co-well-powered* if for any $A \in |\mathbf{K}|$ there exists a *set* of morphisms $E \subseteq \mathbf{E}$ such that for every morphism $e \in \mathbf{E}$ with source A there exist a morphism $e' \in E$ and an isomorphism i such that $e = e';i$. □

Definition 3.3.11 (Quasi-variety). A collection of objects $Q \subseteq |\mathbf{K}|$ is a *quasi-variety* if it is closed under subobjects and products of non-empty sets of objects in Q. □

Lemma 3.3.12 (Initiality lemma). *Assume that* $\mathbf{K}$ *has an initial object, is* $\mathbf{E}$*-co-well-powered, and any set of objects in* $\mathbf{K}$ *has a product. Then any non-empty quasi-variety in* $\mathbf{K}$ *(considered as the corresponding full subcategory of* $\mathbf{K}$*) has an initial object which is reachable in* $\mathbf{K}$*.*

Proof. Let $Q \subseteq |\mathbf{K}|$ be a non-empty collection of objects closed under subobjects and products of non-empty sets. Let Q_r be a *set* of reachable objects in Q such that every reachable object in Q is isomorphic to an element of Q_r (such a set exists since $\mathbf{K}$ is $\mathbf{E}$-co-well-powered). The reachable subobject of the product of Q_r (which is unique up to isomorphism) is a reachable initial object in Q. □

It is now easy to check that in the context of Example 3.3.3 every class of Σ-algebras definable by a set of Σ-equations is a non-empty quasi-variety, and hence Lemma 3.3.12 indeed directly implies Theorem 2.5.14.

We conclude this section with two examples of categories naturally equipped with a notion of reachability which is an instance of the general concept introduced above.

Example 3.3.13. Recall Definitions 2.7.30 and 2.7.31 of partial Σ-algebras and Σ-homomorphisms between them. For any signature Σ, define the category of partial Σ-algebras, $\mathbf{PAlg}(\Sigma)$, as follows:

Objects of $\mathbf{PAlg}(\Sigma)$: partial Σ-algebras;
Morphisms of $\mathbf{PAlg}(\Sigma)$: weak Σ-homomorphisms.

Define also the subcategory $\mathbf{PAlg}_{\mathbf{str}}(\Sigma)$ of partial Σ-algebras with *strong* homomorphisms between them, as follows:

Objects of $\mathbf{PAlg}_{\mathbf{str}}(\Sigma)$: partial Σ-algebras;
Morphisms of $\mathbf{PAlg}_{\mathbf{str}}(\Sigma)$: strong Σ-homomorphisms.

The category $\mathbf{PAlg}(\Sigma)$ of partial Σ-algebras has a factorisation system $\langle \mathbf{PE}_\Sigma, \mathbf{PM}_\Sigma \rangle$, where $\mathbf{PE}_\Sigma$ is the collection of all epimorphisms in $\mathbf{PAlg}(\Sigma)$ and $\mathbf{PM}_\Sigma$ is the collection of all monomorphisms in $\mathbf{PAlg}(\Sigma)$ that are strong Σ-homomorphisms.

Exercise. Characterise epimorphisms in $\mathbf{PAlg}(\Sigma)$ (they are not surjective in general) and prove that $\langle \mathbf{PE}_\Sigma, \mathbf{PM}_\Sigma \rangle$ is indeed a factorisation system for $\mathbf{PAlg}(\Sigma)$. Check then that factorisation of a strong Σ-homomorphism in $\langle \mathbf{PE}_\Sigma, \mathbf{PM}_\Sigma \rangle$ consists of strong Σ-homomorphisms. Conclude that strong homomorphisms in $\mathbf{PE}_\Sigma$ and $\mathbf{PM}_\Sigma$, respectively, form a factorization system for $\mathbf{PAlg}_{\mathbf{str}}(\Sigma)$. □

Example 3.3.14. For any signature Σ, define the category of continuous Σ-algebras, $\mathbf{CAlg}(\Sigma)$, as follows:

Objects of $\mathbf{CAlg}(\Sigma)$: *continuous Σ-algebras*, which are just like ordinary (total) Σ-algebras, except that their carriers are required to be complete partial orders and their operations are continuous functions (cf. Exercise 3.1.9);

Morphisms of $\mathbf{CAlg}(\Sigma)$: *continuous Σ-homomorphisms*: a continuous Σ-homomorphism from a continuous Σ-algebra A to a continuous Σ-algebra B is a Σ-homomorphism $h:A \to B$ which is continuous as a function between complete partial orders. We say that h is *full* if it reflects the ordering, i.e., for all $a, a' \in |A|_s$, $h(a) \leq_B h(a')$ implies $a \leq_A a'$.

The category $\mathbf{CAlg}(\Sigma)$ of continuous Σ-algebras has a factorisation system $\langle \mathbf{CE}_\Sigma, \mathbf{CM}_\Sigma \rangle$, where $\mathbf{CM}_\Sigma$ is the collection of all full monomorphisms in $\mathbf{CAlg}(\Sigma)$ and $\mathbf{CE}_\Sigma$ is the collection of all *strongly dense* epimorphisms in $\mathbf{CAlg}(\Sigma)$. A continuous Σ-homomorphism $h:A \to B$ is strongly dense if B has no proper continuous subalgebra which contains the set-theoretic image of $|A|$ under h. (Note that the expected notion of a continuous subalgebra is determined by the chosen collection of factorisation monomorphisms $\mathbf{CM}_\Sigma$.) This is equivalent to the requirement that every element of $|B|$ is the least upper bound of a countable chain of least upper bounds of countable chains of $\ldots$ of elements in the set-theoretic image of $|A|$ under h. Consequently, given a strongly dense continuous homomorphism $h:A \to B$, every element of $|B|$ is the least upper bound of a subset (not necessarily a chain though) of the set-theoretic image of $|A|$ under h, which yields the key argument to show that $\mathbf{CAlg}(\Sigma)$ is $\mathbf{CE}_\Sigma$-co-well-powered.

Exercise. Prove that $\langle \mathbf{CE}_\Sigma, \mathbf{CM}_\Sigma \rangle$ is indeed a factorisation system for $\mathbf{CAlg}(\Sigma)$. Also, try to construct an example of an epimorphism in $\mathbf{CAlg}(\Sigma)$ which is not strongly dense. □

Exercise 3.3.15. Characterise reachable algebras in $\mathbf{PAlg}(\Sigma)$ and in $\mathbf{CAlg}(\Sigma)$. Instantiate the facts listed in Theorem 3.3.8 to these categories. □

3.4 Functors and natural transformations

As explained in the introduction to this chapter, for category theorists it is tantamount to heresy to consider objects in the absence of morphisms between them. Up to now we have departed from this dogma in our study of categories themselves; in the previous sections of this chapter we have worked with categories without introducing any notion of a morphism between them. We hasten here to correct this lapse: morphisms between categories are *functors*, to be introduced in this section. And by way of atonement we will also introduce *natural transformations*, which are morphisms between functors.

3.4.1 Functors

A category consists of a collection of objects and a collection of morphisms with structure given by the choice of sources and targets of morphism, by the definition of composition and by the identities that are assumed to exist. As in other standard cases of collections with additional structure, morphisms between categories are maps between the collections of objects and morphisms, respectively, that preserve this structure.

Definition 3.4.1 (Functor). A *functor* $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ from a category **K1** to a category **K2** consists of:

- a function $\mathbf{F}_{Obj}:|\mathbf{K1}| \to |\mathbf{K2}|$; and
- for each $A,B \in |\mathbf{K1}|$, a function $\mathbf{F}_{A,B}:\mathbf{K1}(A,B) \to \mathbf{K2}(\mathbf{F}_{Obj}(A),\mathbf{F}_{Obj}(B))$

such that:

- **F** preserves identities: $\mathbf{F}_{A,A}(id_A) = id_{\mathbf{F}_{Obj}(A)}$ for all objects $A \in |\mathbf{K}|$; and
- **F** preserves composition: for all morphisms $f : A \to B$ and $g : B \to C$ in **K1**, $\mathbf{F}_{A,C}(f;g) = \mathbf{F}_{A,B}(f);\mathbf{F}_{B,C}(g)$. □

Notation. We use **F** to refer to both $\mathbf{F}_{Obj}$ and $\mathbf{F}_{A,B}$ for all $A,B \in |\mathbf{K1}|$. □

In the literature, functors as defined above are sometimes referred to as *covariant* functors. A *contravariant* functor is then defined in the same way except that it "reverses the direction of morphisms", i.e., a contravariant functor $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ maps a **K1**-morphism $f:A \to B$ to a **K2**-morphism $\mathbf{F}(f):\mathbf{F}(B) \to \mathbf{F}(A)$. Even though we will use this terminology sometimes, no new formal definition is required: a contravariant functor from **K1** to **K2** is a (covariant) functor from $\mathbf{K1}^{op}$ to **K2** (cf. e.g. Examples 3.4.7 and 3.4.29 below).

Example 3.4.2 (Identity functor). A functor $\mathbf{Id_K}:\mathbf{K} \to \mathbf{K}$ is defined in the obvious way. □

Example 3.4.3 (Inclusion functor). If **K1** is a subcategory of **K2** then the inclusion $\mathbf{I}:\mathbf{K1} \hookrightarrow \mathbf{K2}$ is a functor. □

Example 3.4.4 (Constant functor). For any $A \in |\mathbf{K2}|$, $\mathbf{C}_A:\mathbf{K1} \to \mathbf{K2}$ is a functor, where $\mathbf{C}_A(B) = A$ for any $B \in |\mathbf{K1}|$ and $\mathbf{C}_A(f) = id_A$ for any **K1**-morphism f. □

Example 3.4.5 (Opposite functor). For any functor $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$, there is a functor $\mathbf{F}^{op}:\mathbf{K1}^{op} \to \mathbf{K2}^{op}$ which is the "same" as **F**, but is considered between the opposite categories. □

Example 3.4.6 (Powerset functor). $\mathcal{P}:\mathbf{Set} \to \mathbf{Set}$ is a functor, where $\mathcal{P}(X) = \{Y \mid Y \subseteq X\}$ for any set X, and for any function $f:X \to X'$, $\mathcal{P}(f):\mathcal{P}(X) \to \mathcal{P}(X')$ is defined by $\mathcal{P}(f)(Y) = \{f(y) \mid y \in Y\}$. □

Example 3.4.7 (Contravariant powerset functor). $\mathcal{P}_{-1}\colon \mathbf{Set}^{op} \to \mathbf{Set}$ is a functor, where $\mathcal{P}_{-1}(X) = \{Y \mid Y \subseteq X\}$ for any set X, and for any morphism $f\colon X \to X'$ in $\mathbf{Set}^{op}$ (i.e., any function $f\colon X' \to X$), $\mathcal{P}_{-1}(f)\colon \mathcal{P}_{-1}(X) \to \mathcal{P}_{-1}(X')$ is defined by $\mathcal{P}_{-1}(f)(Y) = \{x' \in X' \mid f(x') \in Y\}$. □

Example 3.4.8 (Sequence functor). $\mathbf{Seq}\colon \mathbf{Set} \to \mathbf{Mon}$ is a functor, where **Mon** is the category of monoids with monoid homomorphisms as morphisms. For any set $X \in |\mathbf{Set}|$, $\mathbf{Seq}(X) = \langle X^*, \hat{}, \varepsilon \rangle$, where X^* is the set of all finite sequences of elements from X, $\hat{}$ is sequence concatenation, and ε is the empty sequence. Then, for any function $f\colon X \to Y$, $\mathbf{Seq}(f)\colon \mathbf{Seq}(X) \to \mathbf{Seq}(Y)$ is the homomorphism defined by $\mathbf{Seq}(f)(x_1 \ldots x_n) = f(x_1) \ldots f(x_n)$. □

Example 3.4.9 (Reduct functor). For any signature morphism $\sigma\colon \Sigma \to \Sigma'$, $\_|_\sigma\colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$ is a functor that takes each Σ'-algebra A' to its σ-reduct $A'|_\sigma \in |\mathbf{Alg}(\Sigma)|$ and each Σ'-homomorphisms h' to its σ-reduct $h'|_\sigma$ (cf. Definitions 1.5.4 and 1.5.8). □

Example 3.4.10 (Forgetful functor). Let $\Sigma = \langle S, \Omega \rangle$ be a signature. Then $|\_|\colon \mathbf{Alg}(\Sigma) \to \mathbf{Set}^S$ is the functor that takes each Σ-algebra $A \in |\mathbf{Alg}(\Sigma)|$ to its S-sorted carrier set $|A| \in |\mathbf{Set}^S|$ and each Σ-homomorphism to its underlying S-sorted function. (The functor $|\_|$ should really be decorated with a subscript identifying the signature Σ — we hope that leaving it out will not confuse the reader.) These special reduct functors $|\_|$ will be referred to as *forgetful functors*.

More generally, the term "forgetful functor" is used to refer to any functor that, intuitively, forgets the structure of objects in a category, mapping any structured object to its underlying unstructured set of elements. Thus, in addition to examples that exactly fit the above definition (like the functor mapping any monoid to the set of its elements) this also covers examples like the functor that maps any topological space to the set of its points and the functor that forgets the metric of a metric space. □

Example 3.4.11 (Term algebra). For any signature $\Sigma = \langle S, \Omega \rangle$, there is a functor $T_\Sigma\colon \mathbf{Set}^S \to \mathbf{Alg}(\Sigma)$ that maps any S-sorted set X to the term algebra $T_\Sigma(X)$, and any S-sorted function $f\colon X \to Y$ to the unique Σ-homomorphism $f^\#\colon T_\Sigma(X) \to T_\Sigma(Y)$ that extends f. □

Exercise 3.4.12. For any signature Σ and set Φ of Σ-equations, define the *quotient functor* $\_/\Phi\colon \mathbf{Alg}(\Sigma) \to \mathbf{Alg}(\Sigma)$ such that for any Σ-algebra A, A/Φ is the quotient of A by the least congruence $\simeq$ on A generated by Φ, that is, such that $t_A(v) \simeq t'_A(v)$ for each Σ-equation $\forall X \bullet t = t'$ in Φ and valuation $v\colon X \to |A|$. Make sure that what you define is a functor! □

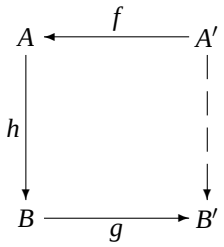
Exercise 3.4.13. For any signature Σ, define the *restriction functor* $\mathbf{R}_\Sigma\colon \mathbf{Alg}(\Sigma) \to \mathbf{Alg}(\Sigma)$ such that for any Σ-algebra A, $\mathbf{R}_\Sigma(A)$ is the reachable subalgebra of A.

More generally: let $\mathbf{K}$ be an arbitrary category with an initial object and a factorisation system, and let $\mathbf{K}_R$ be the full subcategory of $\mathbf{K}$ determined by the collection of all reachable objects in $\mathbf{K}$ (cf. Section 3.3). Define a functor $\mathbf{R}_\mathbf{K}\colon \mathbf{K} \to \mathbf{K}_R$ that maps any $A \in |\mathbf{K}|$ to the (unique up to isomorphism) reachable subobject of A. □

Example 3.4.14 (Projection functor). For any two categories **K1** and **K2**, the projection functors $\Pi_{\mathbf{K1}}\colon \mathbf{K1} \times \mathbf{K2} \to \mathbf{K1}$ and $\Pi_{\mathbf{K2}}\colon \mathbf{K1} \times \mathbf{K2} \to \mathbf{K2}$ are defined by $\Pi_{\mathbf{K1}}(\langle A, B\rangle) = A$ and $\Pi_{\mathbf{K1}}(\langle f, g\rangle) = f$, and $\Pi_{\mathbf{K2}}(\langle A, B\rangle) = B$ and $\Pi_{\mathbf{K2}}(\langle f, g\rangle) = g$. □

Example 3.4.15 (Hom-functor). Let **K** be a locally small category. $\mathbf{Hom}\colon \mathbf{K}^{op} \times \mathbf{K} \to \mathbf{Set}$ is a functor, where $\mathbf{Hom}(\langle A, B\rangle) = \mathbf{K}(A, B)$ and

$$\mathbf{Hom}(\underbrace{\langle f\colon A' \to A, g\colon B \to B'\rangle}_{\in \mathbf{K}^{op} \times \mathbf{K}(\langle A,B\rangle, \langle A',B'\rangle)})(\underbrace{h\colon A \to B}_{\in \mathbf{Hom}(\langle A,B\rangle)}) = \underbrace{f;h;g}_{\in \mathbf{Hom}(\langle A',B'\rangle)} .$$



□

Exercise 3.4.16 (Exponent functor). For any set X define a functor $[\_ \to X]\colon \mathbf{Set}^{op} \to \mathbf{Set}$ mapping any set to the set of all functions from it to X. That is, for any set $Y \in |\mathbf{Set}|$, $[Y \to X]$ is the set of all functions from Y to X and then for any morphism $f\colon Y \to Y'$ in $\mathbf{Set}^{op}$, which is a function $f\colon Y' \to Y$ in **Set**, $[f \to X]\colon [Y \to X] \to [Y' \to X]$ is defined by pre-composition with f as follows: $[f \to X](g) = f;g$. □

Example 3.4.17 (Converting partial functions to total functions). Let **Pfn** be the category of sets with partial functions and let $\mathbf{Set}_\perp$ be the subcategory of **Set** having sets containing a distinguished element $\perp$ as objects and $\perp$-preserving functions as morphisms. Then $\mathbf{Tot}\colon \mathbf{Pfn} \to \mathbf{Set}_\perp$ converts partial functions to total functions by using $\perp$ to represent "undefined" as follows:

- $\mathbf{Tot}(X) = X \uplus \{\perp\}$
- $\mathbf{Tot}(f)(x) = \begin{cases} f(x) & \text{if } f(x) \text{ is defined} \\ \perp & \text{otherwise} \end{cases}$

Exercise. Notice that strictly speaking the above definition is not well-formed: according to the definition of disjoint union, if X is non-empty then $X \not\subseteq X \uplus \{\perp\}$; thus, given a partial function $f\colon X \to Y$, $\mathbf{Tot}(f)$ as defined above need not be a function from $\mathbf{Tot}(X)$ to $\mathbf{Tot}(Y)$. Restate this definition formally, using explicit injections $\iota_1\colon X \to X \uplus \{\perp\}$ and $\iota_2\colon \{\perp\} \to X \uplus \{\perp\}$ for each set X. □

Example 3.4.18 (Converting partial algebras to total algebras). The same "totalisation" idea as used in the above Example 3.4.17 yields a totalisation functor $\mathbf{Tot}_\Sigma\colon \mathbf{PAlg}_{\mathbf{str}}(\Sigma) \to \mathbf{Alg}(\Sigma)$, for each signature Σ, mapping partial Σ-algebras and their strong homomorphisms to total Σ-algebras and their homomorphisms (cf. Definitions 2.7.30 and 2.7.31, and Example 3.3.13).

Let $\Sigma = \langle S, \Omega\rangle \in |\mathbf{AlgSig}|$. $\mathbf{Tot}_\Sigma\colon \mathbf{PAlg}_{\mathbf{str}}(\Sigma) \to \mathbf{Alg}(\Sigma)$ is defined as follows:

- For any partial Σ-algebra $A \in |\mathbf{PAlg}_{\mathbf{str}}(\Sigma)|$, $\mathbf{Tot}_\Sigma(A) \in |\mathbf{Alg}(\Sigma)|$ is the Σ-algebra whose carriers are obtained from the corresponding carriers of A by adding a distinguished element $\bot$, and whose operations are obtained from the operations of A by making the result $\bot$ for arguments on which the latter are undefined, that is:
  - for each sort name $s \in S$, $|\mathbf{Tot}_\Sigma(A)|_s = |A|_s \uplus \{\bot\}$; and
  - for each operation name $f : s_1 \times \ldots \times s_n \to s$ in Σ, $f_{\mathbf{Tot}_\Sigma(A)}$ is the function which yields $\bot$ if any of its arguments is $\bot$, and for $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$,

$$f_{\mathbf{Tot}_\Sigma(A)}(a_1,\ldots,a_n) = \begin{cases} f_A(a_1,\ldots,a_n) & \text{if } f_A(a_1,\ldots,a_n) \text{ is defined} \\ \bot & \text{otherwise.} \end{cases}$$

- For any strong Σ-homomorphism $h : A \to B$ (which is a family of *total* functions between the corresponding carriers of A and B), $\mathbf{Tot}_\Sigma(h) : \mathbf{Tot}_\Sigma(A) \to \mathbf{Tot}_\Sigma(B)$ is (the family of functions in) h extended to map $\bot$ to $\bot$.

Exercise. Check that for any strong Σ-homomorphism $h : A \to B$, $\mathbf{Tot}_\Sigma(h) : \mathbf{Tot}_\Sigma(A) \to \mathbf{Tot}_\Sigma(B)$ is indeed a Σ-homomorphism. Can you extend $\mathbf{Tot}_\Sigma$ to *weak* Σ-homomorphisms between partial algebras? □

Exercise 3.4.19. Do the above functors map monomorphisms to monomorphisms? Do they map epimorphisms to epimorphisms? What about isomorphisms? (Co)limits? (Co)cones? Anything else you can think of? □

Definition 3.4.20 (Diagram translation). Given a functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ and a diagram D in **K1**, the *translation of D by* $\mathbf{F}$ is defined as the diagram $\mathbf{F}(D)$ in **K2** with the same underlying graph as D and with the labels of D translated by $\mathbf{F}$:

- $G(\mathbf{F}(D)) = G(D)$;
- for each $n \in |G(D)|_{node}$, $\mathbf{F}(D)_n = \mathbf{F}(D_n)$; and
- for each $e \in |G(D)|_{edge}$, $\mathbf{F}(D)_e = \mathbf{F}(D_e)$. □

Exercise 3.4.21 (Diagrams as functors). A diagram D in **K** corresponds to a functor from the category $\mathbf{Path}(G(D))$ of paths in the underlying graph of D to **K**. Formalise this. HINT: Given a diagram D, define a functor that maps each path $e_1 \ldots e_n$ in $G(D)$ to $D_{e_1}; \ldots; D_{e_n}$. Do not forget the case where $n = 0$.

Then, anticipating Definition 3.4.27, define the translation of a diagram by a functor in terms of functor composition. □

Definition 3.4.22 (Functor continuity and cocontinuity). A functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ is (*finitely*) *continuous* if it preserves the existing limits of all (finite) diagrams in **K1**, that is, if for any (finite) diagram D in **K1**, **F** maps any limiting cone over D to a limiting cone over $\mathbf{F}(D)$.

A functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ is (*finitely*) *cocontinuous* if it preserves the existing colimits of all (finite) diagrams in **K1**, that is, if for any (finite) diagram D in **K1**, **F** maps any colimiting cocone over D to a colimiting cocone over $\mathbf{F}(D)$. □

Exercise 3.4.23. Assuming that **K1** is (finitely) complete, use Exercise 3.2.49 to show that a functor $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ is (finitely) continuous if and only if it preserves (finite) products and equalisers.

Similarly, show that $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ is finitely continuous if and only if it preserves terminal objects and all pullbacks, and it is continuous if and only if it preserves terminal objects and all wide pullbacks. HINT: Exercises 3.2.48 and 3.2.50).

Dually, give similar characterisation of (finitely) cocontinuous functors, for instance as those that preserve (finite) coproducts and coequalisers. □

Exercise 3.4.24. Given a set X, show that the functor $[\_\to X]:\mathbf{Set}^{op} \to \mathbf{Set}$ from Exercise 3.4.16 is continuous. HINT: Use Exercise 3.4.23: relying on the explicit constructions of (co)products and (co)equalisers in **Set**, show that the functor maps any coproduct (disjoint union) of sets $\langle X_n \rangle_{n\in N}$ to a product of sets of functions $[X_n \to X]$, $n \in N$, and a coequaliser of functions $f,g:X_1 \to X_2$ to an equaliser of (pre-composition) functions $(f;\_),(g;\_):[X_2 \to X] \to [X_1 \to X]$.

You may also want to similarly check which of the examples of functors given above are (finitely) (co)continuous. □

Exercise 3.4.25. Consider a category **K** with a terminal object $1 \in |\mathbf{K}|$. Given any functor $\mathbf{F}:\mathbf{K} \to \mathbf{K}'$, check that **F** determines a functor $\mathbf{F}_{\downarrow 1}:\mathbf{K} \to \mathbf{K}'\downarrow\mathbf{F}(1)$ from **K** to the slice category of $\mathbf{K}'$-objects over $\mathbf{F}(1)$ (Definition 3.1.28), where for any object $A \in |\mathbf{K}|$, $\mathbf{F}_{\downarrow 1}(A) = \mathbf{F}(!_A)$, with $!_A:A \to 1$ being the unique morphism from A to 1, and $\mathbf{F}_{\downarrow 1}$ coincides with **F** on morphisms.

Suppose now that **K** has all pullbacks (so that it is finitely complete) and **F** preserves them (but we do not require **F** to preserve the terminal object, so it does not have to be finitely continuous). Show that $\mathbf{F}_{\downarrow 1}:\mathbf{K} \to \mathbf{K}'\downarrow\mathbf{F}(1)$ is finitely continuous. HINT: Recall Exercise 3.2.51. By the discussion there, since **F** preserves pullbacks, **F** maps products in **K**, which are pullback of morphisms to 1, to pullbacks in $\mathbf{K}'$ of morphisms to $\mathbf{F}(1)$ — and these are essentially products in $\mathbf{K}'\downarrow\mathbf{F}(1)$. Moreover, by the construction, $\mathbf{F}_{\downarrow 1}$ preserves the terminal object, and the conclusion follows by Exercise 3.4.23.

Similarly, show that if **K** has all wide pullbacks (so that it is complete) and **F** preserves them then $\mathbf{F}_{\downarrow 1}:\mathbf{K} \to \mathbf{K}'\downarrow\mathbf{F}(1)$ is continuous. □

Exercise 3.4.26. Recall the definition of the category $\mathbf{T}_{\Sigma,\Phi}$, the algebraic theory generated by a set Φ of equations over a signature Σ (cf. Exercise 3.1.15). Show that those functors from $\mathbf{T}^{op}_{\Sigma,\varnothing}$ to **Set** that preserve finite products (where products in $\mathbf{T}^{op}_{\Sigma,\Phi}$, that is coproducts in $\mathbf{T}_{\Sigma,\Phi}$, are given by concatenation of sequences of sort names, cf. Exercise 3.2.18, and products in **Set** are given by the Cartesian product) are in a bijective correspondence with Σ-algebras in $|\mathbf{Alg}(\Sigma)|$. Generalise this correspondence further to product-preserving functors from $\mathbf{T}^{op}_{\Sigma,\Phi}$ to **Set** and Σ-algebras in $Mod_\Sigma(\Phi)$. □

Definition 3.4.27 (Functor composition). The category **Cat** (the category of all categories) is defined as follows:

Objects of **Cat**: categories[5];
Morphisms of **Cat**: functors;
Composition in **Cat**: If $\mathbf{F}\colon \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G}\colon \mathbf{K2} \to \mathbf{K3}$ are functors, then $\mathbf{F};\mathbf{G}\colon \mathbf{K1} \to \mathbf{K3}$ is a functor defined as follows: $(\mathbf{F};\mathbf{G})_{Obj} = \mathbf{F}_{Obj};\mathbf{G}_{Obj}$ and $(\mathbf{F};\mathbf{G})_{A,B} = \mathbf{F}_{A,B};\mathbf{G}_{\mathbf{F}(A),\mathbf{F}(B)}$ for all $A, B \in \mathbf{K1}$. □

Example 3.4.28. In the following we will often use the functor $|\_|\colon \mathbf{Cat} \to \mathbf{Set}$[6] which for any category $\mathbf{K} \in |\mathbf{Cat}|$ yields the collection $|\mathbf{K}|$ of the objects of this category and for each functor $\mathbf{F}\colon \mathbf{K} \to \mathbf{K}'$ yields its object part $|\mathbf{F}| = \mathbf{F}_{Obj}\colon |\mathbf{K}| \to |\mathbf{K}'|$. □

Example 3.4.29. $\mathbf{Alg}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ is a functor, where:

- for any $\Sigma \in |\mathbf{AlgSig}|$, $\mathbf{Alg}(\Sigma)$ is the category of Σ-algebras; and
- for any morphism $\sigma\colon \Sigma \to \Sigma'$ in **AlgSig**, $\mathbf{Alg}(\sigma)$ is the reduct functor $\_|_\sigma\colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$. □

Exercise 3.4.30. Define a functor $\mathbf{Alg}^{der}\colon (\mathbf{AlgSig}^{der})^{op} \to \mathbf{Cat}$ so that $\mathbf{Alg}^{der}(\Sigma) = \mathbf{Alg}(\Sigma)$ for any signature $\Sigma \in |\mathbf{AlgSig}^{der}|$, and for any derived signature morphism δ, $\mathbf{Alg}^{der}(\delta)$ is the δ-reduct as sketched in Definition 1.5.16 and Exercise 1.5.17. □

Exercise 3.4.31. Define the category **Poset** (objects: partially-ordered sets; morphisms: order-preserving functions). Define the functor from **Poset** to **Cat** that maps a partially-ordered set to the corresponding (preorder) category (cf. Example 3.1.3) and an order-preserving function to the corresponding functor. □

Exercise 3.4.32. Characterise isomorphisms in **Cat**. Show that product categories are products in **Cat**. What are terminal objects, pullbacks and equalisers in **Cat**? Conclude that **Cat** is complete. HINT: Use constructions analogous to those in **Set**, as summarised in Exercise 3.2.53. □

Exercise 3.4.33. Prove that $\mathbf{Alg}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ (cf. Example 3.4.29) is continuous, that is, that it maps colimits in the category **AlgSig** of signatures to limits in the category **Cat** of all categories.

HINT: By Exercise 3.4.23 it is enough to show that **Alg** maps coproducts of signatures to products of the corresponding categories of algebras and coequalisers of signature morphisms to equalisers of the corresponding reduct functors.

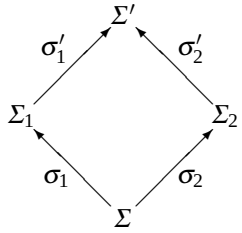
(*Coproducts*): Recall that by Exercise 3.2.16, a coproduct of signatures is in fact their disjoint union. Now, it is easy to see that an algebra over a disjoint union of a family of signatures may be identified with a tuple of algebras over the signatures in the family. Since a similar fact holds for homomorphisms, the rest of the proof in this case is straightforward (cf. Exercise 3.4.32). Notice that this argument covers the coproduct of the empty family of signatures as well.

[5] To be cautious about the set-theoretic foundations here, we should rather say: *small* categories.

[6] Again, we should restrict attention to small categories here. Alternatively, in place of **Set** we could use the category of all discrete categories, inheriting all of the foundational problems of **Cat**.

(*Coequalisers*): Recall (cf. Exercise 3.2.24) that a coequaliser of two signature morphisms $\sigma, \sigma' : \Sigma \to \Sigma'$ is the natural projection $p : \Sigma' \to (\Sigma'/\equiv)$, where $\equiv$ is the least equivalence relation on Σ' such that $\sigma(x) \equiv \sigma'(x)$ for all sort and operation names x in Σ (this is just a sketch of the construction). Notice now that $(\Sigma'/\equiv)$-algebras correspond exactly to those Σ'-algebras that have identical components $\sigma(x)$ and $\sigma'(x)$ for all sort and operation names x in Σ, or equivalently, to those algebras $A' \in |\mathbf{Alg}(\Sigma')|$ for which $A'|_\sigma = A'|_{\sigma'}$. Moreover, the correspondence is given by the functor $\_|_p : \mathbf{Alg}(\Sigma'/\equiv) \to \mathbf{Alg}(\Sigma')$. Since a similar fact holds for homomorphisms, it is straightforward now to prove that $\_|_p = \mathbf{Alg}(p)$ is an equaliser of $\_|_\sigma = \mathbf{Alg}(\sigma)$ and $\_|_{\sigma'} = \mathbf{Alg}(\sigma')$ (cf. Exercises 3.4.32 and 3.2.22). □

Exercise 3.4.34 (Amalgamation Lemma for algebras). Consider a pushout in the category **AlgSig** of signatures:



Conclude from Exercise 3.4.33 above that for any Σ_1-algebra A_1 and Σ_2-algebra A_2 such that $A_1|_{\sigma_1} = A_2|_{\sigma_2}$, there exists a unique Σ'-algebra A' such that $A'|_{\sigma'_1} = A_1$ and $A'|_{\sigma'_2} = A_2$.

Similarly, for any two homomorphisms $h_1 : A_{11} \to A_{12}$ in $\mathbf{Alg}(\Sigma_1)$ and $h_2 : A_{21} \to A_{22}$ in $\mathbf{Alg}(\Sigma_2)$ such that $h_1|_{\sigma_1} = h_2|_{\sigma_2}$, there exists a unique Σ'-homomorphism $h' : A'_1 \to A'_2$ such that $h'|_{\sigma'_1} = h_1$ and $h'|_{\sigma'_2} = h_2$. □

Example 3.4.35. Recall Example 3.2.35 of a simple pushout of algebraic signatures. Let $N \in |\mathbf{Alg}(\Sigma\mathrm{NAT})|$ be the standard model of natural numbers. Build $N_1 \in |\mathbf{Alg}(\Sigma\mathrm{NAT}_{\mathit{fib}})|$ by adding to N the interpretation of the operation *fib* as the standard Fibonacci function, and $N_2 \in |\mathbf{Alg}(\Sigma\mathrm{NAT}_{\mathit{mult}})|$ by adding to N the interpretation of the operation *mult* as multiplication. By construction we have $N_1|_{\Sigma\mathrm{NAT}} = N = N_2|_{\Sigma\mathrm{NAT}}$ and so N_1 and N_2 amalgamate to a unique algebra $N' \in |\mathbf{Alg}(\Sigma\mathrm{NAT}_{\mathit{fib},\mathit{mult}})|$ such that $N'|_{\Sigma\mathrm{NAT}_{\mathit{fib}}} = N_1$ and $N'|_{\Sigma\mathrm{NAT}_{\mathit{mult}}} = N_2$. Clearly, N' is the only expansion of N that defines *fib* as the Fibonacci function (as N_1 does) and *mult* as multiplication (as N_2 does). □

Exercise 3.4.36. Define initial objects and coproducts in **Cat**. (HINT: This is easy.) Try to define coequalisers and then pushouts in **Cat**. (HINT: This is difficult.) □

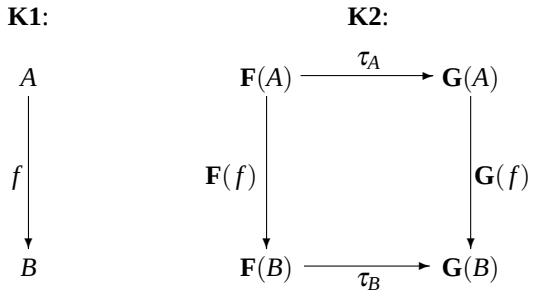
¡¡¡¡¡¡¡ c342.tex ======= ¿¿¿¿¿¿¿ 1.15

3.4.2 Natural transformations

Let $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G}:\mathbf{K1} \to \mathbf{K2}$ be two functors with common source and target categories.

A transformation from **F** to **G** should map the results of **F** to the results of **G**. This means, that it should consists of a family of morphisms in **K2**, one **K2**-morphism from $\mathbf{F}(A)$ to $\mathbf{G}(A)$ for each **K1**-object A. An extra requirement to impose is that this family should be compatible with the application of **F** and **G** to **K1**-morphisms, as formalised by the following definition:

Definition 3.4.37 (Natural transformation). A *natural transformation* from **F** to **G**, $\tau:\mathbf{F} \to \mathbf{G}$[7], is a family $\langle \tau_A:\mathbf{F}(A) \to \mathbf{G}(A)\rangle_{A\in|\mathbf{K1}|}$ of **K2**-morphisms such that for any $A,B \in |\mathbf{K1}|$ and **K1**-morphism $f:A \to B$ the following diagram commutes:



(this property is often referred to as the *naturality* of the family τ).

Furthermore, τ is a *natural isomorphism* if for all $A \in |\mathbf{K1}|$, τ_A is iso (in **K2**). □

Example 3.4.38. The identity transformation $id_{\mathbf{F}}:\mathbf{F} \to \mathbf{F}$, where $(id_{\mathbf{F}})_A = id_{\mathbf{F}(A)}$, is a natural isomorphism.

For any morphism $f:A \to B$ in a category **K2** and for any category **K1**, there is a constant natural transformation $\mathbf{c}_f:\mathbf{C}_A \to \mathbf{C}_B$ between the constant functors $\mathbf{C}_A,\mathbf{C}_B:\mathbf{K1} \to \mathbf{K2}$ (cf. Example 3.4.4) defined by $(\mathbf{c}_f)_o = f$ for all objects $o \in |\mathbf{K1}|$. □

Example 3.4.39. The family of singleton functions $sing\_set:\mathbf{Id}_{\mathbf{Set}} \to \mathcal{P}$, where for any set X, $sing\_set_X:X \to \mathcal{P}(X)$ is defined by $sing\_set_X(a) = \{a\}$, is a natural transformation.

Let $(\_)^* = \mathbf{Seq};|\_|:\mathbf{Set} \to \mathbf{Set}$ be the functor given as the composition of $\mathbf{Seq}:\mathbf{Set} \to \mathbf{Mon}$ (Example 3.4.8) with the forgetful functor $|\_|:\mathbf{Mon} \to \mathbf{Set}$ mapping any monoid to its underlying carrier set. The family of singleton functions $sing\_seq:\mathbf{Id}_{\mathbf{Set}} \to (\_)^*$, where for any set X, $sing\_seq_X:X \to X^*$ is defined by $sing\_seq_X(a) = a$ (*sing_seq* maps any element to the singleton sequence consisting of this element only) is a natural transformation. □

[7] Some authors would use a dotted or double arrow here, writing $\tau:\mathbf{F} \dot{\to} \mathbf{G}$ or $\tau:\mathbf{F} \Rightarrow \mathbf{G}$, respectively. We prefer to use the same symbol for all morphisms, and also for natural transformations, since they are morphisms in certain categories, see Definition 3.4.60 below.

Exercise 3.4.40. Consider the functor $(\_)^*:\mathbf{Set}\to\mathbf{Set}$ mapping any set X to the set X^* of sequences over X (cf. Example 3.4.39 above). Show that the following families of functions (indexed by sets $X\in|\mathbf{Set}|$) yield natural transformations from $(\_)^*$ to $(\_)^*$:

- for each $k\geq 0$, for $n\geq 0$ and $x_1,\ldots,x_n\in X$, $\mathit{stutter}^k_X(x_1\ldots x_n)=\underbrace{x_1\ldots x_1}_{k\text{ times}}\ldots\underbrace{x_n\ldots x_n}_{k\text{ times}}$;
- for each $k\geq 0$, for $n\geq 0$ and $x_1,\ldots,x_n\in X$, $\mathit{repeat}^k_X(x_1\ldots x_n)=\underbrace{x_1\ldots x_n\ldots x_1\ldots x_n}_{k\text{ times}}$;
- for $n\geq 0$ and $x_1,\ldots,x_n\in X$, $\mathit{reverse}_X(x_1\ldots x_n)=x_n\ldots x_1$;
- for $n\geq 0$ and $x_1,\ldots,x_{2n+1}\in X$, $\mathit{odds}_X(x_1x_2x_3\ldots x_{2n})=x_1x_3\ldots x_{2n-1}$ and $\mathit{odds}_X(x_1x_2x_3\ldots x_{2n+1})=x_1x_3\ldots x_{2n+1}$.

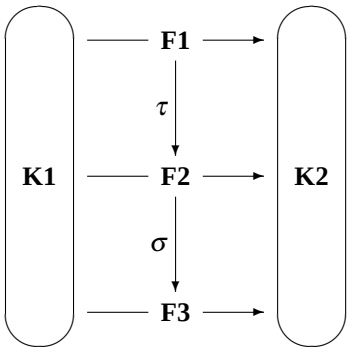
Check which of these functions also yield natural transformations from **Seq** to **Seq** (where $\mathbf{Seq}:\mathbf{Set}\to\mathbf{Mon}$, cf. Example 3.4.8).

The above examples indicate a close link between polymorphic functions as encountered in functional programming languages (like Standard ML [MTHM97] or Haskell [Pey03]) and natural transformations between functors representing polymorphic types. This property, often referred to as "parametric polymorphism" (as opposed to "ad hoc polymorphism") can be explored to derive some propeties of polymorphic functions directly from their types [Wad89]. □

Exercise 3.4.41. Recall (Exercise 3.4.26) the correspondence between product-preserving functors from $\mathbf{T}^{op}_{\Sigma,\Phi}$ to **Set** and Σ-algebras in $|\mathbf{Mod}(\Sigma,\Phi)|$. Show that this correspondence extends to morphisms: each Σ-homomorphism between algebras gives rise to a natural transformation between the corresponding functors, and vice versa, each natural transformation between such functors determines a homomorphism between the corresponding algebras. HINT: To prove that this yields a bijective correspondence, first use the naturality condition for product projections to show that for any natural transformation $\tau:\mathbf{F}\to\mathbf{G}$ between product-preserving functors $\mathbf{F},\mathbf{G}:\mathbf{T}^{op}_{\Sigma,\Phi}\to\mathbf{Set}$, any sequence $s_1\ldots s_n$ of sort names (an object in $\mathbf{T}_{\Sigma,\Phi}$) and any $\langle a_1,\ldots,a_n\rangle\in\mathbf{F}(s_1\ldots s_n)$, $\tau_{s_1\ldots s_n}(\langle a_1,\ldots,a_n\rangle)=\langle\tau_{s_1}(a_1),\ldots,\tau_{s_n}(a_n)\rangle$. □

Natural transformations have been introduced as morphisms between functors. The obvious thing to do next is to define composition of natural tranformations. Traditionally, two different composition operations for natural transformations are introduced: *vertical* and *horizontal* composition. The former is a straightforward composition of natural transformations between parallel functors. The latter is somewhat more involved; in a sense, it shows how natural transformations "accumulate" when functors are composed.

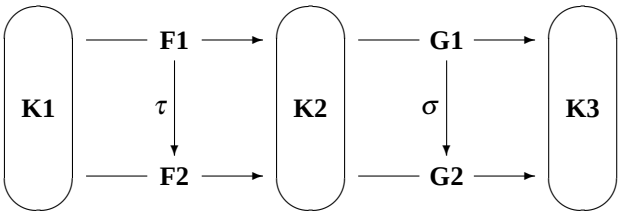
Definition 3.4.42 (Vertical composition). Let $\mathbf{F1},\mathbf{F2},\mathbf{F3}:\mathbf{K1}\to\mathbf{K2}$ be three functors with common source and target categories. Let $\tau:\mathbf{F1}\to\mathbf{F2}$ and $\sigma:\mathbf{F2}\to\mathbf{F3}$ be natural transformations:



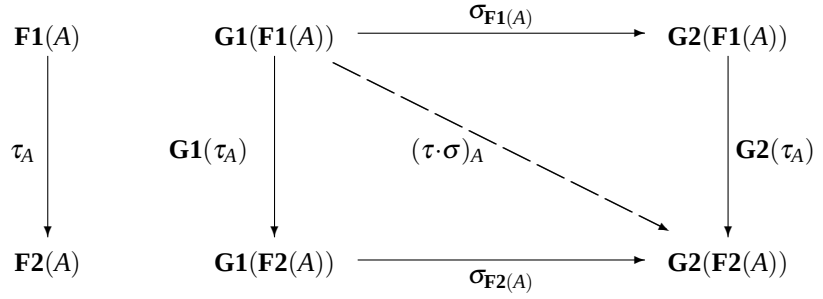
Then the *vertical composition* of τ and σ, $\tau;\sigma:\mathbf{F1}\to\mathbf{F3}$, is defined by $(\tau;\sigma)_A = \tau_A;\sigma_A$ (in **K2**) for all $A\in|\mathbf{K1}|$. □

Exercise 3.4.43. Prove that $\tau;\sigma$ is indeed a natural transformation. □

Definition 3.4.44 (Horizontal composition). Let $\mathbf{F1},\mathbf{F2}:\mathbf{K1}\to\mathbf{K2}$ and $\mathbf{G1},\mathbf{G2}:\mathbf{K2}\to\mathbf{K3}$ be two pairs of parallel functors. Let $\tau:\mathbf{F1}\to\mathbf{F2}$ and $\sigma:\mathbf{G1}\to\mathbf{G2}$ be natural transformations:

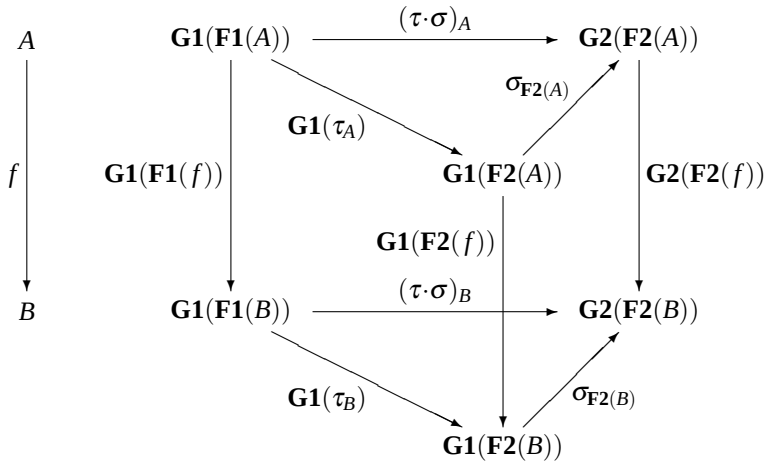


Then the *horizontal composition* of τ and σ, $\tau\cdot\sigma:\mathbf{F1};\mathbf{G1}\to\mathbf{F2};\mathbf{G2}$, is defined by $(\tau\cdot\sigma)_A = \mathbf{G1}(\tau_A);\sigma_{\mathbf{F2}(A)} = \sigma_{\mathbf{F1}(A)};\mathbf{G2}(\tau_A)$ (in **K3**) for all $A\in|\mathbf{K1}|$:



□

Exercise 3.4.45. Prove that the above diagram commutes, and so $(\tau\cdot\sigma)_A$ is well-defined. Then prove that $\tau\cdot\sigma$ is indeed a natural transformation. HINT:



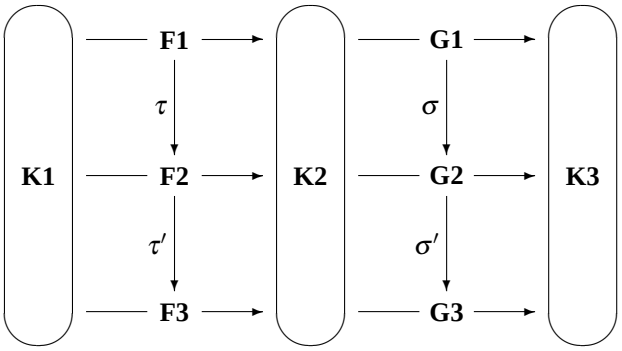
□

Definition 3.4.46 (Multiplication by a functor). A special case of the horizontal composition of natural transformations is the *multiplication* of a natural transformation by a functor. Under the assumptions of Definition 3.4.44, we define:

- $\tau \cdot \mathbf{G1} = \tau \cdot id_{\mathbf{G1}} : \mathbf{F1};\mathbf{G1} \to \mathbf{F2};\mathbf{G1}$, or more explicitly: $(\tau \cdot \mathbf{G1})_A = \mathbf{G1}(\tau_A)$ for $A \in |\mathbf{K1}|$;
- $\mathbf{F1} \cdot \sigma = id_{\mathbf{F1}} \cdot \sigma : \mathbf{F1};\mathbf{G1} \to \mathbf{F1};\mathbf{G2}$, or more explicitly: $(\mathbf{F1} \cdot \sigma)_A = \sigma_{\mathbf{F1}(A)}$ for $A \in |\mathbf{K1}|$. □

Exercise 3.4.47. Show that $\tau \cdot \sigma = (\tau \cdot \mathbf{G1});(\mathbf{F2} \cdot \sigma) = (\mathbf{F1} \cdot \sigma);(\tau \cdot \mathbf{G2})$. □

Exercise 3.4.48 (Interchange law). Consider any categories **K1**, **K2**, **K3**, functors $\mathbf{F1}, \mathbf{F2}, \mathbf{F3} : \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G1}, \mathbf{G2}, \mathbf{G3} : \mathbf{K2} \to \mathbf{K3}$, and natural transformations $\tau : \mathbf{F1} \to \mathbf{F2}$, $\tau' : \mathbf{F2} \to \mathbf{F3}$, $\sigma : \mathbf{G1} \to \mathbf{G2}$, and $\sigma' : \mathbf{G2} \to \mathbf{G3}$:



Show that $(\tau;\tau') \cdot (\sigma;\sigma') = (\tau \cdot \sigma);(\tau' \cdot \sigma')$. □

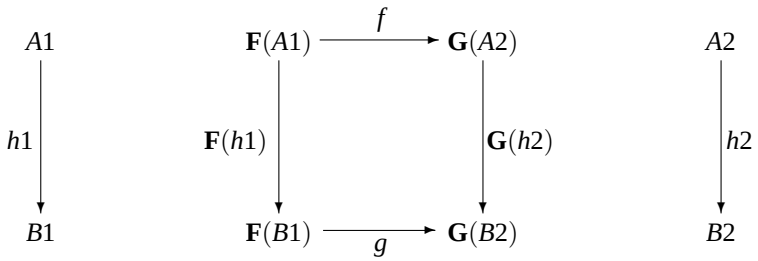
3.4.3 Constructing categories, revisited

3.4.3.1 Comma categories

Definition 3.4.49 (Comma category). Let $\mathbf{F}:\mathbf{K1}\to\mathbf{K}$ and $\mathbf{G}:\mathbf{K2}\to\mathbf{K}$ be two functors with a common target category. The *comma category* $(\mathbf{F},\mathbf{G})$ is defined by:

Objects of $(\mathbf{F},\mathbf{G})$: triples $\langle A1, f, A2\rangle$, where $A1\in|\mathbf{K1}|$, $A2\in|\mathbf{K2}|$ and $f:\mathbf{F}(A1)\to\mathbf{G}(A2)$ is a morphism in $\mathbf{K}$;

Morphisms of $(\mathbf{F},\mathbf{G})$: a morphism from $\langle A1, f, A2\rangle$ to $\langle B1, g, B2\rangle$ is a pair $\langle h1, h2\rangle$ of morphisms where $h1:A1\to B1$ (in **K1**) and $h2:A2\to B2$ (in **K2**) such that (the middle part of) the following diagram commutes:



Composition in $(\mathbf{F},\mathbf{G})$: $\langle h1,h2\rangle;\langle h1',h2'\rangle=\langle h1;h1',h2;h2'\rangle$. □

Exercise 3.4.50. Construct the category $\mathbf{K}^{\to}$ of $\mathbf{K}$-morphisms and the category $\mathbf{K}{\downarrow}A$ of $\mathbf{K}$-objects over $A\in|\mathbf{K}|$ as comma categories (cf. Definitions 3.1.27 and 3.1.28). HINT: Consider categories $(\mathbf{Id_K},\mathbf{Id_K})$ and $(\mathbf{Id}_K,\mathbf{C}^{\mathbf{1}}_A)$, where $\mathbf{Id_K}$ is the identity functor on $\mathbf{K}$ and $\mathbf{C}^{\mathbf{1}}_A:\mathbf{1}\to\mathbf{K}$ is a constant functor from the terminal category $\mathbf{1}$. □

Example 3.4.51. Another way of presenting the category **Graph** is as the comma category $(\mathbf{Id_{Set}},\mathbf{CP})$, where $\mathbf{CP}:\mathbf{Set}\to\mathbf{Set}$ is the Cartesian product functor defined by $\mathbf{CP}(X)=X\times X$ and $\mathbf{CP}(f:X\to Y)\langle x1,x2\rangle=\langle f(x1),f(x2)\rangle$.

To see this, write an object in $|(\mathbf{Id_{Set}},\mathbf{CP})|$ as $\langle E,\langle source:E\to N, target:E\to N\rangle, N\rangle$. □

Exercise 3.4.52. Another way to present the category of signatures **AlgSig** is as the comma category $(\mathbf{Id_{Set}},(\_)^+)$, where $(\_)^+:\mathbf{Set}\to\mathbf{Set}$ is the functor which for any set $X\in|\mathbf{Set}|$ yields the set X^+ of all finite non-empty sequences of elements from X.

First, complete the definition of the functor $(\_)^+$. Then, notice that $X^+=X^*\times X$ and hence an object in $|(\mathbf{Id_{Set}},(\_)^+)|$ may be written as $\langle\Omega,\langle arity:\Omega\to S^*, sort:\Omega\to S\rangle, S\rangle$. Indicate now why the category defined is almost, but not quite, the same as the category **AlgSig** of signatures (cf. Exercise 3.4.75 below). □

Exercise 3.4.53. Prove that if **K1** and **K2** are (finitely) complete categories, $\mathbf{F}:\mathbf{K1}\to\mathbf{K}$ is a functor, and $\mathbf{G}:\mathbf{K2}\to\mathbf{K}$ is a (finitely) continuous functor, then the comma category $(\mathbf{F},\mathbf{G})$ is (finitely) complete. Moreover, the obvious projections from $(\mathbf{F},\mathbf{G})$ to **K1** and **K2**, respectively, are (finitely) continuous. HINT: To construct a limit

of a diagram in $(\mathbf{F},\mathbf{G})$, start by building limits of the projections of the diagram to **K1** and **K2**, respectively, and then use the continuity property of **G** to complete the construction of the limit object in $(\mathbf{F},\mathbf{G})$. If the notation in the proof gets too heavy, use Exercise 3.2.49 and spell the details out for the construction of products and equalisers.

Check that this construction of limits in $(\mathbf{F},\mathbf{G})$ works for diagrams of any given shape: if **K1** and **K2** have limits of diagrams of a given shape, and **G** preserves them, then $(\mathbf{F},\mathbf{G})$ has limits of diagrams of this shape, and the projection functors preserve them.

State and prove the analogous facts about cocompleteness of $(\mathbf{F},\mathbf{G})$. HINT: Clearly, appropriate colimits must exist in **K1** and **K2**, but unlike with limits, it is **F** that must preserve them. □

Exercise 3.4.54. Use Exercises 3.4.50 and 3.4.53 to show that if **K** is a (finitely) complete category then so is the category $\mathbf{K}^{\rightarrow}$ of morphisms in **K**.

Then, without looking at Exercise 3.2.51, use Exercises 3.4.50 and 3.4.53 to prove that if a category **K** has limits of all (finite) non-empty connected diagrams then so does the slice category $\mathbf{K}{\downarrow}A$ of its objects over $A \in |\mathbf{K}|$, and that the obvious forgetful functor from $\mathbf{K}{\downarrow}A$ to **K** preserves these limits. Notice though that this does not generalise to arbitrary (finite) limits that exist in $\mathbf{K}{\downarrow}A$ if **K** is (finitely) complete by Exercise 3.2.51.

Check that your proof shows a stronger fact: without assuming the existence of any limits in **K**, the forgetful functor from $\mathbf{K}{\downarrow}A$ to **K** *creates* limits of all non-empty connected diagrams, that is: for any such diagram $D_{\downarrow A}$ in $\mathbf{K}{\downarrow}A$, if its projection D to **K** has a limit in **K** then there is a unique cocone on $D_{\downarrow A}$ in $\mathbf{K}{\downarrow}A$ that projects to this limit, and this cocone is a limit of $D_{\downarrow A}$ in $\mathbf{K}{\downarrow}A$. □

Exercise 3.4.55. Show that if **K** has all pullbacks and a terminal object (so, it is finitely complete) and a functor $\mathbf{F}\colon\mathbf{K}\to\mathbf{K}'$ preserves pullbacks, then **F** also preserves the limits of all finite non-empty connected diagrams. HINT: Put together Exercises 3.4.25 and 3.4.54.

Similarly, show that if **K** has all wide pullbacks and a terminal object (so, it is complete) and a functor $\mathbf{F}\colon\mathbf{K}\to\mathbf{K}'$ preserves wide pullbacks, then **F** also preserves the limits of all non-empty connected diagrams. □

3.4.3.2 Indexed categories

We frequently need to deal not just with a single category, but rather with a family of categories, "parameterised" by a certain collection of indices. The categories of S-sorted sets (one for each set S) and the categories of Σ-algebras (one for each signature Σ) are typical examples. A crucial property here is that all the categories in such a family are defined in a uniform way, and consequently any change of an index induces a smooth translation between the corresponding component categories. In typical examples, the translation goes in the opposite direction than the change of index, which leads to the following definition:

Definition 3.4.56 (Indexed category). An *indexed category* (over an *index category* **Ind**) is a functor $\mathbf{C}:\mathbf{Ind}^{op}\to\mathbf{Cat}$. □

Example 3.4.57. $\mathbf{Alg}:\mathbf{AlgSig}^{op}\to\mathbf{Cat}$ is an indexed category (cf. Example 3.4.29). □

Definition 3.4.58 (Grothendieck construction). Every indexed category $\mathbf{C}:\mathbf{Ind}^{op}\to\mathbf{Cat}$ gives rise to a *flattened* category $\mathbf{Flat}(\mathbf{C})$ defined as follows:

Objects of $\mathbf{Flat}(\mathbf{C})$: pairs $\langle i,A\rangle$ for all $i\in|\mathbf{Ind}|$ and $A\in|\mathbf{C}(i)|$;
Morphisms of $\mathbf{Flat}(\mathbf{C})$: a morphism from $\langle i,A\rangle$ to $\langle j,B\rangle$ is a pair $\langle\sigma,f\rangle:\langle i,A\rangle\to\langle j,B\rangle$, where $\sigma:i\to j$ is an **Ind**-morphism and $f:A\to\mathbf{C}(\sigma)(B)$ is a $\mathbf{C}(i)$-morphism;
Composition in $\mathbf{Flat}(\mathbf{C})$: $\langle\sigma,f\rangle;\langle\sigma',f'\rangle=\langle\sigma;\sigma',f;\mathbf{C}(\sigma)(f')\rangle$. □

Exercise 3.4.59. Show that if **Ind** is complete, $\mathbf{C}(i)$ are complete for all $i\in|\mathbf{Ind}|$, and $\mathbf{C}(\sigma)$ are continuous for all $\sigma\in\mathbf{Ind}$, then $\mathbf{Flat}(\mathbf{C})$ is complete.

HINT: Given a diagram in the flattened category $\mathbf{Flat}(\mathbf{C})$, first consider its obvious projection on the index category **Ind**. Since **Ind** is complete, this has a limit $l\in|\mathbf{Ind}|$. Using the functors assigned by **C** to the projection morphism of the limit, "translate" all the nodes and edges of the diagram to the category $\mathbf{C}(l)$, thus obtaining a diagram in $\mathbf{C}(l)$. Since $\mathbf{C}(l)$ is complete, it has a limit. Check that the projection morphisms of the limit of the diagram constructed in **Ind** when paired with the corresponding projection morphisms of the limit of the diagram in $\mathbf{C}(l)$ form the limit of the original diagram in $\mathbf{Flat}(\mathbf{C})$.

To make the construction manageable, consider only products and equalisers: this is sufficient by Exercise 3.2.49. □

3.4.3.3 Functor categories

Definition 3.4.60 (Functor category). Let **K1** and **K2** be categories[8]. The *functor category* [**K1**→**K2**] is defined by:

Objects of [**K1**→**K2**]: functors from **K1** to **K2**;
Morphisms of [**K1**→**K2**]: natural transformations;
Composition in [**K1**→**K2**]: vertical composition. □

Exercise 3.4.61. Define the category $\mathbf{Set}^S$ of S-sorted sets as a functor category. □

Exercise 3.4.62. For any category **K**, define its morphism category $\mathbf{K}^{\to}$ as the category of functors [**2**→**K**]. □

Exercise 3.4.63. Let **K1** and **K2** be categories. Show that if **K2** is (finitely) complete then so is the functor category [**K1**→**K2**]. State and show the dual fact as well.
HINT: The limit of any diagram in [**K1**→**K2**] may be constructed "pointwise", for

[8] To be cautious about set-theoretic foundations, one may want to assume that **K1** is small.

each object in $|\mathbf{K1}|$ separately. More precisely, using Exercise 3.2.49 to simplify the notational burden: consider any family of functors $\langle \mathbf{F}_n : \mathbf{K1} \to \mathbf{K2} \rangle_{n \in N}$. For each $X \in |\mathbf{K1}|$, let $\mathbf{Q}(X) \in |\mathbf{K2}|$ with projections $(\pi_n)_X : \mathbf{Q}(X) \to \mathbf{F}_n(X)$, $n \in N$, be a product of $\langle \mathbf{F}_n(X) \rangle_{n \in N}$ in $\mathbf{K2}$. Check that there is a unique way to extend $\mathbf{Q}$ to a functor $\mathbf{Q} : \mathbf{K1} \to \mathbf{K2}$ so that all $\pi_n : \mathbf{Q} \to \mathbf{F}_n$, $n \in N$, become natural transformations. Show that $\mathbf{Q}$ with projections $\langle \pi_n \rangle_{n \in N}$ is a product of $\langle \mathbf{F}_n : \mathbf{K1} \to \mathbf{K2} \rangle_{n \in N}$ in $[\mathbf{K1} \to \mathbf{K2}]$. Then proceed similarly for equalisers: consider functors $\mathbf{F}, \mathbf{F}' : \mathbf{K1} \to \mathbf{K2}$ and natural transformations $\tau_1, \tau_2 : \mathbf{F} \to \mathbf{F}'$. For each $X \in |\mathbf{K1}|$, let $\tau_X : \mathbf{Q}(X) \to \mathbf{F}(X)$ be an equaliser of $(\tau_1)_X, (\tau_2)_X : \mathbf{F}(X) \to \mathbf{F}'(X)$ in $\mathbf{K2}$. This yields a unique functor $\mathbf{Q} : \mathbf{K1} \to \mathbf{K2}$ such that $\tau : \mathbf{Q} \to \mathbf{F}$ is a natural transformation, which is an equaliser of τ_1, τ_2 in $[\mathbf{K1} \to \mathbf{K2}]$. □

Exercise 3.4.64. Let $\mathbf{K1}$, $\mathbf{K1}'$ and $\mathbf{K2}$ be categories. Show how any functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K1}'$ induces a functor $(\mathbf{F};\_) : [\mathbf{K1}' \to \mathbf{K2}] \to [\mathbf{K1} \to \mathbf{K2}]$. Relying on the construction outlined in Exercise 3.4.63 and assuming that $\mathbf{K2}$ is (finitely) complete, show that this functor is (finitely) continuous.

Prove also that this yields a functor $[\_ \to \mathbf{K2}] : \mathbf{Cat}^{op} \to \mathbf{Cat}$[9] (cf. Exercise 3.4.16). □

Exercise 3.4.65. For any category $\mathbf{K}$, define a category $\mathbf{Funct}(\mathbf{K})$ of *functors into* $\mathbf{K}$ as follows:

Objects of $\mathbf{Funct}(\mathbf{K})$: functors $\mathbf{F} : \mathbf{K}' \to \mathbf{K}$ into $\mathbf{K}$;
Morphisms of $\mathbf{Funct}(\mathbf{K})$: a morphism from $\mathbf{F} : \mathbf{K1} \to \mathbf{K}$ to $\mathbf{G} : \mathbf{K2} \to \mathbf{K}$ is a pair $\langle \Phi, \rho \rangle$, where $\Phi : \mathbf{K1} \to \mathbf{K2}$ is a functor and $\rho : \mathbf{F} \to \Phi;\mathbf{G}$ is a natural transformation (between functors from $\mathbf{K1}$ to $\mathbf{K}$);
Composition in $\mathbf{Funct}(\mathbf{K})$: $\langle \Phi, \rho \rangle ; \langle \Phi', \rho' \rangle = \langle \Phi;\Phi', \rho;(\Phi \cdot \rho') \rangle$.

Show how the category $\mathbf{Funct}(\mathbf{K})$ arises by the flattening construction of Definition 3.4.58 for the functor $[\_ \to \mathbf{K}]$ as defined in the previous exercise.[10] □

Exercise 3.4.66. Show that if $\mathbf{K}$ is a (finitely) complete category then the category $\mathbf{Funct}(\mathbf{K})$ of functors into $\mathbf{K}$ is (finitely) complete as well. HINT: You may construct the limits in $\mathbf{Funct}(\mathbf{K})$ directly, perhaps using Exercise 3.2.49. Alternatively, rely on the construction of $\mathbf{Funct}(\mathbf{K})$ by flattening (Definition 3.4.58) for the functor $[\_ \to \mathbf{K}] : \mathbf{Cat}^{op} \to \mathbf{Cat}$ and on Exercise 3.4.59; recall that $\mathbf{Cat}$ is complete by Exercise 3.4.32, for any category $\mathbf{K1}$, $[\mathbf{K1} \to \mathbf{K}]$ is (finitely) complete by Exercise 3.4.63, and for every functor $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$, $(\mathbf{F};\_) : [\mathbf{K2} \to \mathbf{K}] \to [\mathbf{K1} \to \mathbf{K}]$ is (finitely) continuous by Exercise 3.4.64. □

Exercise 3.4.67. Show that if a category $\mathbf{K1}$ has a factorisation system (cf. Section 3.3) than for any category $\mathbf{K2}$, the functor category $[\mathbf{K2} \to \mathbf{K1}]$ has a factorisation system as well.

HINT: Let $\langle \mathbf{E1}, \mathbf{M1} \rangle$ be a factorisation system for $\mathbf{K1}$. Define $\mathbf{E} = \{\varepsilon \in [\mathbf{K2} \to \mathbf{K1}] \mid \varepsilon_A \in \mathbf{E1} \text{ for } a \in |\mathbf{K2}|\}$ and $\mathbf{M} = \{\eta \in [\mathbf{K2} \to \mathbf{K1}] \mid \eta_A \in \mathbf{M1} \text{ for } a \in |\mathbf{K2}|\}$. Now,

[9] Assuming that $\mathbf{K2}$ is small would help to resolve potential foundational problems here.

[10] So, for foundational reasons, one may prefer to keep all categories small around here as well.

to construct an $\langle \mathbf{E}, \mathbf{M} \rangle$-factorisation of a natural transformation $\tau : \mathbf{F} \to \mathbf{G}$ between functors $\mathbf{F}, \mathbf{G} : \mathbf{K2} \to \mathbf{K1}$, first for each object $a \in |\mathbf{K2}|$ obtain an $\langle \mathbf{E1}, \mathbf{M1} \rangle$-factorisation of τ_A, say $\tau_A = \varepsilon_A ; \eta_A$ with $\varepsilon_A \in \mathbf{E1}$ and $\eta_A \in \mathbf{M1}$, and $\varepsilon_A : \mathbf{F}(A) \to \mathbf{H}(A)$, $\eta_A : \mathbf{H}(A) \to \mathbf{G}(A)$ for some $\mathbf{H}(A) \in |\mathbf{K1}|$. Then use the diagonal fill-in lemma (Lemma 3.3.4) to extend the mapping $\mathbf{H} : |\mathbf{K2}| \to |\mathbf{K1}|$ to a functor $\mathbf{H} : \mathbf{K2} \to \mathbf{K1}$ such that $\varepsilon : \mathbf{F} \to \mathbf{H}$ and $\eta : \mathbf{H} \to \mathbf{G}$ are natural transformations. □

3.4.3.4 Equivalence of categories

Definition 3.4.68 (Isomorphic categories). Two categories **K1** and **K2** are *isomorphic* if there are functors $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{F}^{-1} : \mathbf{K2} \to \mathbf{K1}$ such that $\mathbf{F};\mathbf{F}^{-1} = \mathbf{Id}_{\mathbf{K1}}$ and $\mathbf{F}^{-1};\mathbf{F} = \mathbf{Id}_{\mathbf{K2}}$. □

In other words, we say that two categories are isomorphic if they are isomorphic as objects of **Cat**. As with isomorphic objects of other kinds, we will view isomorphic categories as abstractly the same. It turns out, however, that in this case there is a coarser relation which allows us to identify categories which have all the same categorical properties, even though they may not be isomorphic.

Definition 3.4.69 (Equivalent categories). **K1** and **K2** are *equivalent* if there are functors $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G} : \mathbf{K2} \to \mathbf{K1}$ and natural isomorphisms $\tau : \mathbf{Id}_{\mathbf{K1}} \to \mathbf{F};\mathbf{G}$ and $\sigma : \mathbf{G};\mathbf{F} \to \mathbf{Id}_{\mathbf{K2}}$. □

To characterise equivalent categories, we need one more concept:

Definition 3.4.70 (Skeletal category). A category **K** is *skeletal* iff any two isomorphic **K**-objects are identical. A *skeleton of* **K** is any maximal skeletal subcategory of **K**. □

Exercise 3.4.71. Prove that two categories are equivalent iff they have isomorphic skeletons. □

Thus, intuitively, two categories are equivalent if and only if they differ only in the number of isomorphic copies of corresponding objects.

Example 3.4.72. The category **FinSet** of all finite sets is equivalent to its full subcategory of all natural numbers, where any natural number n is defined as the set $\{0, \ldots, n-1\}$ of all natural numbers smaller than n. In fact, the latter is a skeleton of **FinSet**. Similarly, the category **Set** of all sets is equivalent to its full subcategory of all ordinals. □

Exercise 3.4.73. Show that for any signature Σ and set Φ of Σ-equations, the full subcategory of $\mathbf{T}_\Sigma/\Phi$ given by the finite sets of variables is equivalent to the category $\mathbf{T}_{\Sigma,\Phi}$ (cf. Exercises 3.1.14 and 3.1.15). □

Exercise 3.4.74. Let **K1** and **K2** be equivalent categories. Show that if **K1** is (finitely) (co)complete then so is **K2**. □

Exercise 3.4.75. Recall Exercise 3.4.52. As indicated there, categories **AlgSig** and $(\mathbf{Id}_{\mathbf{Set}}, (\_)^+)$ are not isomorphic. Show that they are equivalent. Then, using Exercises 3.4.74 and 3.4.53, conclude from this that **AlgSig** is complete and cocomplete. □

3.5 Adjoints

Recall Facts 1.4.4 and 1.4.10:

Fact 1.4.4. *For any Σ-algebra A and S-sorted function $v: X \to |A|$ there is exactly one Σ-homomorphism $v^\#: T_\Sigma(X) \to A$ which extends v, i.e. such that $v^\#_s(\iota_X(x)) = v_s(x)$ for all $s \in S$, $x \in X_s$, where $\iota_X: X \to |T_\Sigma(X)|$ is the embedding that maps each variable in X to the corresponding term.* □

Fact 1.4.10. *This property defines $T_\Sigma(X)$ up to isomorphism: if B is a Σ-algebra and $\eta: X \to |B|$ is an S-sorted function such that for any Σ-algebra A and S-sorted function $v: X \to |A|$ there is a unique Σ-homomorphism $v^\$: B \to A$ such that $\eta; |v^\$| = v$ then B is isomorphic to $T_\Sigma(X)$.* □

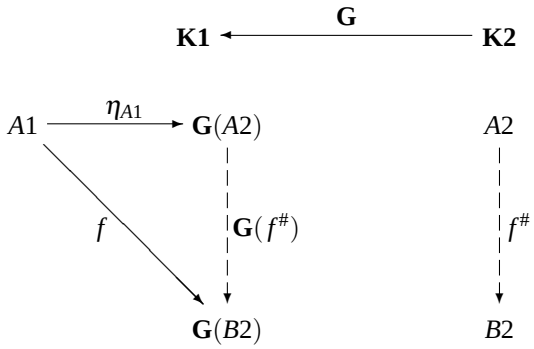
The construction of the algebra of Σ-terms is one example of an *adjoint functor* (it is *left adjoint* to the functor $|\_|: \mathbf{Alg}(\Sigma) \to \mathbf{Set}^{sorts(\Sigma)}$). The general concept of an adjoint functor, to which this section is devoted, has many other important instances. In fact, [Gog91b] goes so far as to say:

> *Any canonical construction from widgets to whatsits is an adjoint of another functor, from whatsits to widgets.*

3.5.1 Free objects

Let **K1** and **K2** be categories, $\mathbf{G}: \mathbf{K2} \to \mathbf{K1}$ be a functor, and $A1$ be an object of **K1**.

Definition 3.5.1 (Free object). A *free object over $A1$ w.r.t.* **G** is a **K2**-object $A2$ together with a **K1**-morphism $\eta_{A1}: A1 \to \mathbf{G}(A2)$ such that for any **K2**-object $B2$ and **K1**-morphism $f: A1 \to \mathbf{G}(B2)$ there is a unique **K2**-morphism $f^\#: A2 \to B2$ such that $\eta_{A1}; \mathbf{G}(f^\#) = f$.



η_{A1} is called the *unit morphism*. □

Example 3.5.2. Let $\Sigma = \langle S, \Omega \rangle$ be an arbitrary signature. Consider the forgetful functor $|\_|: \mathbf{Alg}(\Sigma) \to \mathbf{Set}^S$. Fact 1.4.4 asserts that for any S-sorted set X, the term algebra $T_\Sigma(X)$ with the inclusion $\eta_X: X \hookrightarrow |T_\Sigma(X)|$ is a free object over X w.r.t. $|\_|$. □

Exercise 3.5.3. Define free monoids and the path categories $\mathbf{Path}(G)$ as free objects w.r.t. some obvious functors. Then, look around at the areas of mathematics with which you are familiar for more examples. For instance, check that free groups and discrete topologies, (ideal) completion of partial orders, of ordered algebras, etc. may be defined as free objects w.r.t. some straightforward functors. □

Exercise 3.5.4. Prove that any free object over $A1$ w.r.t. $\mathbf{G}$ is an initial object in the comma category $(\mathbf{C}_{A1}, \mathbf{G})$, where $\mathbf{C}_{A1}: \mathbf{1} \to \mathbf{K1}$ is the constant functor. Conclude that a free object over $A1$ w.r.t. $\mathbf{G}$ is unique up to isomorphism. □

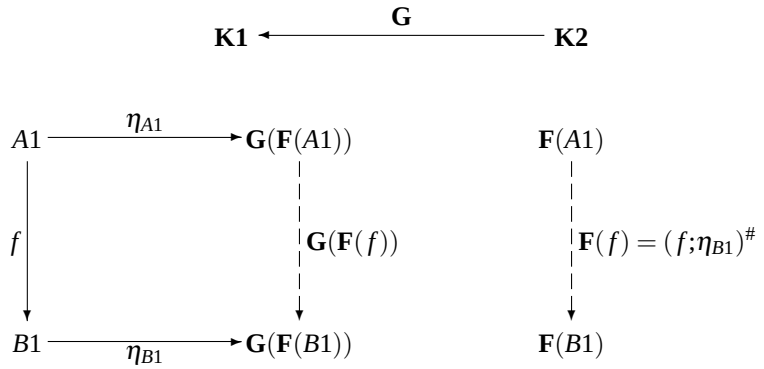
Exercise 3.5.5. Prove that if $A2 \in |\mathbf{K2}|$ is a free object over $A1 \in |\mathbf{K1}|$ w.r.t. $\mathbf{G}: \mathbf{K2} \to \mathbf{K1}$, then for any $B2 \in |\mathbf{K2}|$, $\#: \mathbf{K1}(A1, \mathbf{G}(B2)) \to \mathbf{K2}(A2, B2)$ is a bijection.

Check that one consequence of this is that two morphisms $g, h: A2 \to B2$ coincide (in $\mathbf{K2}$) whenever $\eta_{A1}; \mathbf{G}(g) = \eta_{A1}; \mathbf{G}(h)$ in $\mathbf{K1}$. □

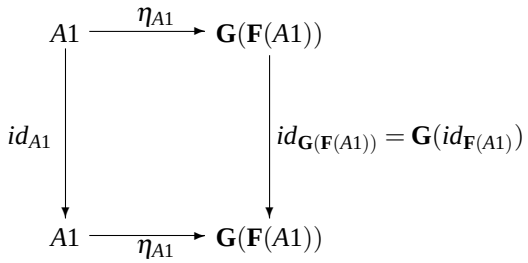
3.5.2 Left adjoints

Let $\mathbf{K1}$ and $\mathbf{K2}$ be categories and $\mathbf{G}: \mathbf{K2} \to \mathbf{K1}$ be a functor. So far we have considered free objects w.r.t. $\mathbf{G}$ one by one, without relating them with each other. One crucial property is that the construction of free objects, if they exist, is functorial.

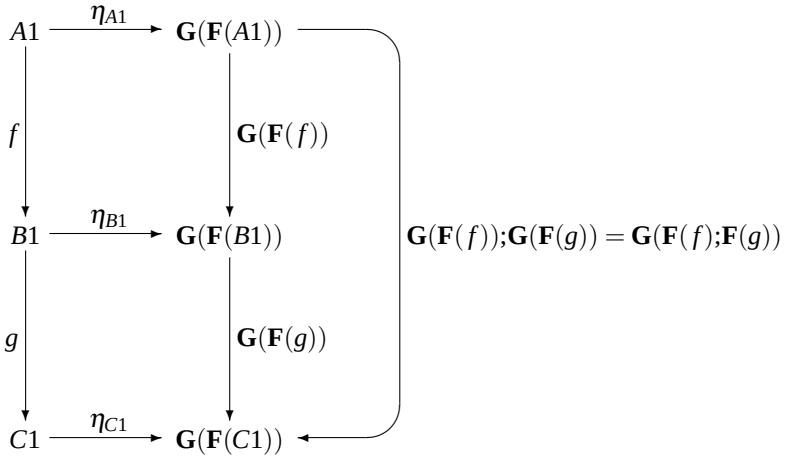
Proposition 3.5.6. *If for any $A1 \in |\mathbf{K1}|$ there is a free object over $A1$ w.r.t. $\mathbf{G}$, say $F(A1) \in |\mathbf{K2}|$ with unit morphism $\eta_{A1}: A1 \to \mathbf{G}(F(A1))$ (in $\mathbf{K1}$), then $A1 \mapsto F(A1)$ and $f \in \mathbf{K1}(A1, B1) \mapsto (f; \eta_{B1})^\# \in \mathbf{K2}(F(A1), F(B1))$ determine a functor $\mathbf{F}: \mathbf{K1} \to \mathbf{K2}$.*



Proof. **F** *preserves identities*: $\mathbf{F}(id_{A1}) = (id_{A1};\eta_{A1})^{\#} = id_{\mathbf{F}(A1)}$ follows from the fact that the following diagram commutes:



F *preserves composition*: Since the following diagram commutes:



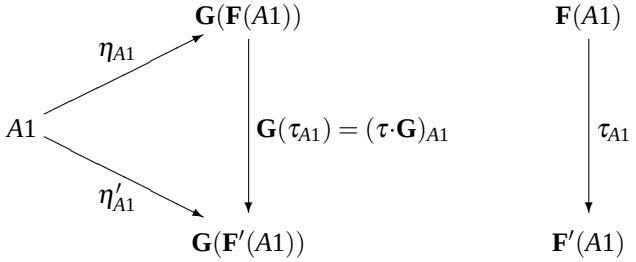
it follows that $\mathbf{F}(f;g) = (f;g;\eta_{C1})^{\#} = \mathbf{F}(f);\mathbf{F}(g)$. □

Exercise 3.5.7. Prove that $\eta : \mathbf{Id_{K1}} \to \mathbf{F};\mathbf{G}$ in Proposition 3.5.6 is a natural transformation. □

Definition 3.5.8 (Left adjoint). Let $\mathbf{F}:\mathbf{K1}\to\mathbf{K2}$ and $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$ be functors and $\eta:\mathbf{Id_{K1}}\to\mathbf{F};\mathbf{G}$ be a natural transformation. $\mathbf{F}$ is *left adjoint to* $\mathbf{G}$ *with unit* η if for any $A1\in|\mathbf{K1}|$, $\mathbf{F}(A1)$ with unit morphism $\eta_{A1}:A1\to\mathbf{G}(\mathbf{F}(A1))$ is a free object over $A1$ w.r.t. $\mathbf{G}$. □

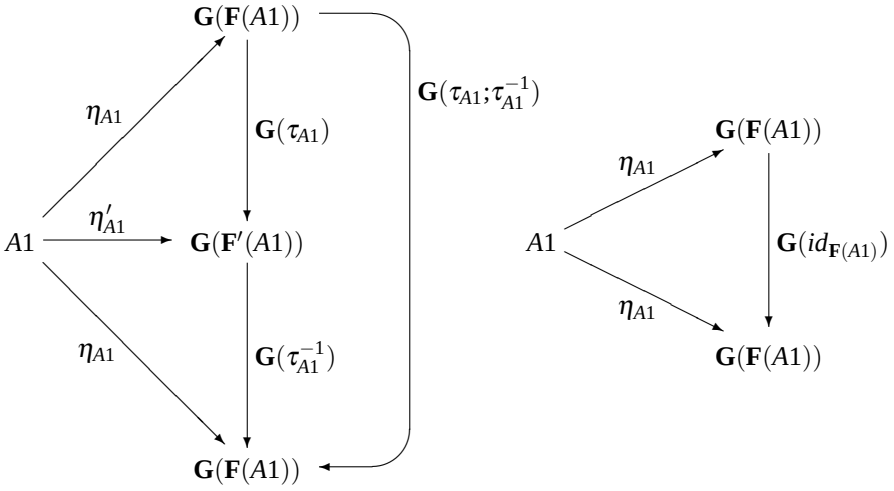
Before we give any examples, let us prove a very important property of left adjoints.

Proposition 3.5.9. *A left adjoint to* $\mathbf{G}$ *is unique up to (natural) isomorphism: if* $\mathbf{F}$ *and* $\mathbf{F}'$ *are left adjoints of* $\mathbf{G}$ *with units* η *and* η' *respectively, then there is a natural isomorphism* $\tau:\mathbf{F}\to\mathbf{F}'$ *such that* $\eta;(\tau\cdot\mathbf{G})=\eta'$.



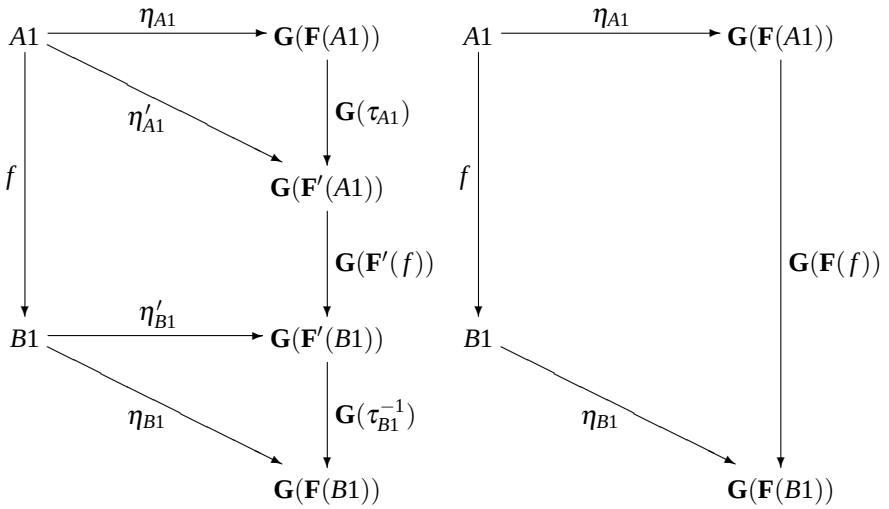
Proof. First notice that for any $f\in\mathbf{K1}(A1,B1)$, $\mathbf{F}(f)=(f;\eta_{B1})^{\#}$ and $\mathbf{F}'(f)=(f;\eta'_{B1})^{\#'}$.

Then, for $A1\in|\mathbf{K1}|$, define $\tau_{A1}=(\eta'_{A1})^{\#}$ and $\tau_{A1}^{-1}=(\eta_{A1})^{\#'}$. Then $\tau_{A1};\tau_{A1}^{-1}=id_{\mathbf{F}(A1)}$ since the following diagrams commute:



and $\tau_{A1}^{-1};\tau_{A1}=id_{\mathbf{F}'(A1)}$ by a similar argument.

Finally, for $f:A1\to B1$ (in **K1**), the following diagrams commute:



Thus, $\mathbf{F}(f) = (f;\eta_{B1})^{\#} = \tau_{A1};\mathbf{F}'(f);\tau_{B1}^{-1}$. This proves that $\mathbf{F}(f);\tau_{B1} = \tau_{A1};\mathbf{F}'(f)$, and hence that $\tau:\mathbf{F} \to \mathbf{F}'$ is natural. □

Example 3.5.10. For any signature $\Sigma = \langle S, \Omega \rangle$, the functor $T_\Sigma:\mathbf{Set}^S \to \mathbf{Alg}(\Sigma)$ is left adjoint to the forgetful functor $|\_|:\mathbf{Alg}(\Sigma) \to \mathbf{Set}^S$ (cf. Examples 3.4.11 and 3.4.9).

The functor $\mathbf{Seq}:\mathbf{Set} \to \mathbf{Mon}$ is left adjoint to the forgetful functor $|\_|:\mathbf{Mon} \to \mathbf{Set}$ which takes a monoid to its underlying set of elements. The unit is $\mathit{sing\_seq}:\mathbf{Id}_{\mathbf{Set}} \to \mathbf{Seq};|\_|$ (cf. Examples 3.4.8 and 3.4.39).

The "free group" functor $\mathbf{F}:\mathbf{Set} \to \mathbf{Grp}$ is left adjoint to the forgetful functor $|\_|:\mathbf{Grp} \to \mathbf{Set}$. Also, the functor taking a set X to the discrete topology on X is left adjoint to the forgetful functor $|\_|:\mathbf{Top} \to \mathbf{Set}$ (cf. Exercise 3.5.3). □

Exercise 3.5.11. Consider any algebraic signature morphism $\sigma:\Sigma \to \Sigma'$. Prove that the reduct functor $\_|_\sigma:\mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$ has a left adjoint.

HINT: Formalise and complete the following construction. For any Σ-algebra A, let $\Sigma(A)$ be an algebraic signature which extends Σ by a constant $a:s$ for each element $a \in |A|_s$, $s \in \mathit{sorts}(\Sigma)$, and let $\Sigma'(A)$ be a similar extension of Σ' by a constant $a:\sigma(s)$ for each $a \in |A|_s$, $s \in \mathit{sorts}(\Sigma)$. Consider the congruence $\equiv_A$ on $T_{\Sigma(A)}$ generated by the identities that hold in A. The congruence $\equiv_A$ may be translated by σ to $\Sigma'(A)$-terms, generating there a congruence $\sigma(\equiv_A)$, and the algebra $T_{\Sigma'(A)}/\sigma(\equiv_A)$ is (almost) the free Σ'-algebra over A.

Consider then a set Φ' of Σ'-equations. Recall that $\mathbf{Mod}(\Sigma', \Phi')$ is the full subcategory of $\mathbf{Alg}(\Sigma')$ with all Σ'-algebras that satisfy Φ' as objects (cf. Example 3.1.19). Prove that the reduct functor $\_|_\sigma:\mathbf{Mod}(\Sigma', \Phi') \to \mathbf{Alg}(\Sigma)$ has a left adjoint.

HINT: In the construction above, close the congruence $\sigma(\equiv_A)$ so that for each equation $\forall X' \bullet t = t'$ in Φ' and substitution $\theta:X' \to |T_{\Sigma'(A)}|$, it identifies the terms $t[\theta]$ and $t'[\theta]$ (cf. Exercise 1.4.9 for the notation used here).

Finally, for any set Φ of Σ-equations such that $\Phi' \models_{\Sigma'} \sigma(\Phi)$, prove that the reduct functor $\_|_\sigma : \mathbf{Mod}(\Sigma', \Phi') \to \mathbf{Mod}(\Sigma, \Phi)$ has a left adjoint.

HINT: This is easy now (Proposition 2.3.13 ensures that the functor is well defined). □

Exercise 3.5.12. Generalise Exercise 3.5.11 to derived signature morphisms, with reduct functors as introduced in Exercise 3.4.30. □

Example 3.5.13. Let $\mathbf{K}$ be a category, and recall that $\mathbf{1}$ is a category containing a single object, say a. Let $\mathbf{F} : \mathbf{1} \to \mathbf{K}$ be left adjoint to $\mathbf{C}_a : \mathbf{K} \to \mathbf{1}$ (note that such a functor $\mathbf{F}$ may not exist). Then $\mathbf{F}(a)$ is an initial object in $\mathbf{K}$. □

Exercise 3.5.14. Let $\Delta : \mathbf{K} \to \mathbf{K} \times \mathbf{K}$ be the "diagonal" functor such that $\Delta(A) = \langle A, A \rangle$ and $\Delta(f : A \to B) = \langle f, f \rangle : \Delta(A) \to \Delta(B)$. Prove that $\mathbf{K}$ has all coproducts iff Δ has a left adjoint. What is the unit? □

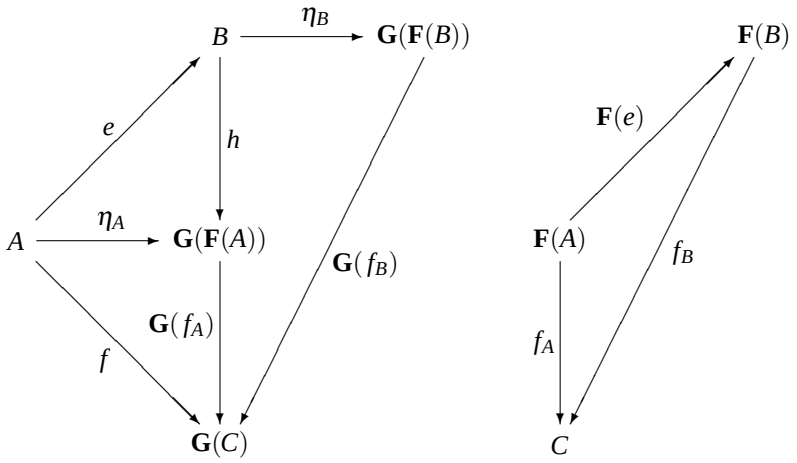
Exercise 3.5.15. Formulate analogous theorems for coequalisers and pushouts and prove them. Show how this may be done for any colimit. □

Exercise 3.5.16. Let $\mathbf{K}$ be a category with an initial object and a factorisation system and let $\mathbf{K}_R$ be its full subcategory of reachable objects. Recall that $\mathbf{R}_\mathbf{K} : \mathbf{K} \to \mathbf{K}_R$ is a functor that maps any object to its reachable subobject (cf. Exercise 3.4.13). Show that the inclusion functor $\mathbf{I} : \mathbf{K}_R \to \mathbf{K}$ is left adjoint to $\mathbf{R}_K$. □

Exercise 3.5.17. Show that left adjoints preserve colimits of diagrams. Do they preserve limits as well? □

Exercise 3.5.18. Let $\mathbf{F} : \mathbf{K2} \to \mathbf{K1}$ be left adjoint to $\mathbf{G} : \mathbf{K1} \to \mathbf{K2}$ with unit $\eta : \mathbf{Id}_\mathbf{K1} \to \mathbf{F};\mathbf{G}$. Consider two objects $A, B \in |\mathbf{K1}|$ and suppose that for some epimorphism $e : A \to B$ there exists a morphism $h : B \to \mathbf{G}(\mathbf{F}(A))$ such that $e;h = \eta_A$. Prove that $\mathbf{F}(e) : \mathbf{F}(A) \to \mathbf{F}(B)$ is an isomorphism.

HINT:



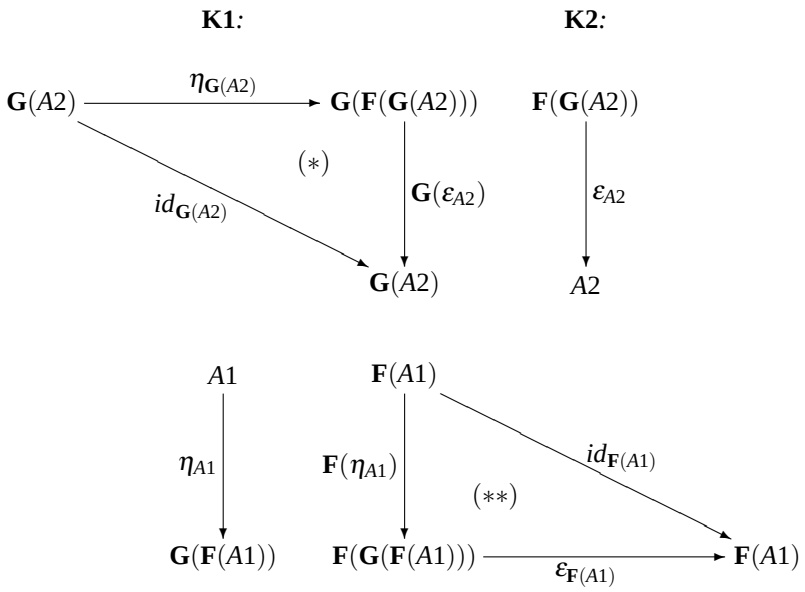
First show that $\mathbf{F}(B)$ with $e;\eta_B : A \to \mathbf{G}(\mathbf{F}(B))$ as the unit morphism is a free object over A w.r.t. $\mathbf{G}$. For this, use the following construction: for any $C \in |\mathbf{K2}|$ and $f : A \to \mathbf{G}(C)$, let $f_B : \mathbf{F}(B) \to C$ be the unique morphism such that $\eta_B;\mathbf{G}(f_B) = h;\mathbf{G}(f_A)$, where in turn $f_A : \mathbf{F}(A) \to C$ is the unique morphism such that $\eta_A;\mathbf{G}(f_A) = f$. Now, f_B satisfies $(e;\eta_B);\mathbf{G}(f_B) = f$ and moreover, it is the only morphism from $\mathbf{F}(B)$ to C with this property (use the fact that e is an epimorphism and the freeness of $\mathbf{F}(B)$ to prove the latter). Then, show the conclusion following the proof of the uniqueness of left adjoints, cf. Proposition 3.5.9. □

3.5.3 Adjunctions

Consider two categories **K1** and **K2** and functors $\mathbf{F} : \mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G} : \mathbf{K2} \to \mathbf{K1}$ such that **F** is left adjoint to **G** with unit $\eta : \mathbf{Id_{K1}} \to \mathbf{F};\mathbf{G}$.

Proposition 3.5.19. *There is a natural transformation $\varepsilon : \mathbf{G};\mathbf{F} \to \mathbf{Id_{K2}}$ such that*

$$(*): \quad (\mathbf{G}\cdot\eta);(\varepsilon\cdot\mathbf{G}) = id_{\mathbf{G}}$$
$$(**): \quad (\eta\cdot\mathbf{F});(\mathbf{F}\cdot\varepsilon) = id_{\mathbf{F}}$$



Proof idea.

- $(*)$ defines $\varepsilon_{A2} : \mathbf{F}(\mathbf{G}(A2)) \to A2$ as $\varepsilon_{A2} = (id_{\mathbf{G}(A2)})^{\#}$.
- *Check naturality*: To show that for all $g : A2 \to B2$ in **K2**, $\varepsilon_{A2};g = \mathbf{F}(\mathbf{G}(g));\varepsilon_{B2}$, it is enough to prove that (in **K1**) $\eta_{\mathbf{G}(A2)};\mathbf{G}(\varepsilon_{A2};g) = \eta_{\mathbf{G}(A2)};\mathbf{G}(\mathbf{F}(\mathbf{G}(g));\varepsilon_{B2})$.
- *Check* $(**)$: To prove that $\mathbf{F}(\eta_{A1});\varepsilon_{\mathbf{F}(A1)} = id_{\mathbf{F}(A1)}$, it is enough to show that (in **K1**) $\eta_{A1};\mathbf{G}(\mathbf{F}(\eta_{A1});\varepsilon_{\mathbf{F}(A1)}) = \eta_{A1};\mathbf{G}(id_{\mathbf{F}(A1)})$. □

Proposition 3.5.20. *Consider functors* $\mathbf{F}:\mathbf{K1}\to\mathbf{K2}$ *and* $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$*, and natural transformations* $\eta:\mathbf{Id_{K1}}\to\mathbf{F};\mathbf{G}$ *and* $\varepsilon:\mathbf{G};\mathbf{F}\to\mathbf{Id_{K2}}$ *such that*

$$\begin{array}{rl}(*): & (\mathbf{G}\cdot\eta);(\varepsilon\cdot\mathbf{G})=id_{\mathbf{G}}\\ (**): & (\eta\cdot\mathbf{F});(\mathbf{F}\cdot\varepsilon)=id_{\mathbf{F}}\end{array}$$

Then $\mathbf{F}$ *is left adjoint to* $\mathbf{G}$ *with unit* η.

Proof. For $A1\in|\mathbf{K1}|$, $B2\in|\mathbf{K2}|$, $f:A1\to\mathbf{G}(B2)$, let $f^{\#}=\mathbf{F}(f);\varepsilon_{B2}:\mathbf{F}(A1)\to B2$.

- $\eta_{A1};\mathbf{G}(f^{\#})=\eta_{A1};\mathbf{G}(\mathbf{F}(f));\mathbf{G}(\varepsilon_{B2})=f;\eta_{\mathbf{G}(B2)};\mathbf{G}(\varepsilon_{B2})=f;id_{\mathbf{G}(B2)}=f$.
- Suppose that for some $g:\mathbf{F}(A1)\to B2$, $\eta_{A1};\mathbf{G}(g)=f$. Then: $f^{\#}=\mathbf{F}(f);\varepsilon_{B2}=\mathbf{F}(\eta_{A1};\mathbf{G}(g));\varepsilon_{B2}=\mathbf{F}(\eta_{A1});\mathbf{F}(\mathbf{G}(g));\varepsilon_{B2}=\mathbf{F}(\eta_{A1});\varepsilon_{\mathbf{F}(A1)};g=g$. □

Definition 3.5.21 (Adjunction). Let **K1** and **K2** be categories. An *adjunction from* **K1** *to* **K2** is a quadruple $\langle\mathbf{F},\mathbf{G},\eta,\varepsilon\rangle$ where $\mathbf{F}:\mathbf{K1}\to\mathbf{K2}$ and $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$ are functors and $\eta:\mathbf{Id_{K1}}\to\mathbf{F};\mathbf{G}$ and $\varepsilon:\mathbf{G};\mathbf{F}\to\mathbf{Id_{K2}}$ are natural transformations such that

$$\begin{array}{rl}(*): & (\mathbf{G}\cdot\eta);(\varepsilon\cdot\mathbf{G})=id_{\mathbf{G}}\\ (**): & (\eta\cdot\mathbf{F});(\mathbf{F}\cdot\varepsilon)=id_{\mathbf{F}}\end{array}$$ □

Fact 3.5.22. *Equivalently, an adjunction may be given as either of the following:*

- *A functor* $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$ *and for each* $A1\in|\mathbf{K1}|$*, a free object over* $A1$ *w.r.t.* $\mathbf{G}$*;*
- *A functor* $\mathbf{G}:\mathbf{K2}\to\mathbf{K1}$ *and its left adjoint.* □

Exercise 3.5.23 (Galois connection). Recall that any partial order gives rise to a corresponding preorder category (cf. Example 3.1.3). Galois connections (Definition 2.3.3) arise as adjunctions between preorder categories:

Consider two partially ordered sets $\langle A,\leq_A\rangle$ and $\langle B,\leq_B\rangle$ and two order-preserving functions $f:A\to B$ and $g:B\to A$ (i.e., for $a,a'\in A$, if $a\leq_A a'$ then $f(a)\leq_B f(a')$ and for $b,b'\in B$, if $b\leq_B b'$ then $g(b)\leq_A g(b')$).

Show that f and g (viewed as functors) form an adjunction between $\langle A,\leq_A\rangle$ and $\langle B,\leq_B\rangle$ (viewed as categories) if and only if for all $a\in A$ and $b\in B$:

$$a\leq_A g(b)\quad\text{iff}\quad f(a)\leq_B b$$

Then show that this is further equivalent to the requirement that:

- $a\leq_A g(f(a))$ for all $a\in A$; and
- $f(g(b))\leq_B b$ for all $b\in B$.

View the Galois connection between sets of equations and classes of algebras on a given signature defined in Section 2.3 (cf. Proposition 2.3.2) as a special case of the above definition. That is, check that for any signature Σ, the function mapping any set of Σ-equations to the class of all Σ-algebras that satisfy this set of equations and the function mapping any class of Σ-algebras to the set of all Σ-equations that hold in this class form an adjunction between the powerset of the set of Σ-equations (ordered by inclusion) and the powerclass of the class of Σ-algebras (ordered by containment).

Then check that the above definition of Galois connection coincides with the more explicit Definition 2.3.3 of a Galois connection between $\langle A, \leq_A \rangle$ and $\langle B, \geq_B \rangle$ (note the opposite order for B). □

Exercise 3.5.24. Dualise the development in this section. Begin with the following definition, dual to Definition 3.5.1:

Definition. Let $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ be a functor and let $A2 \in |\mathbf{K2}|$. A *cofree object over* $A2$ *w.r.t.* $\mathbf{F}$ is a $\mathbf{K1}$-object $A1$ together with a $\mathbf{K2}$-morphism $\varepsilon_{A2}:\mathbf{F}(A1) \to A2$ such that for any $\mathbf{K1}$-object $B1$ and $\mathbf{K2}$-morphism $f:\mathbf{F}(B1) \to A2$ there is a unique $\mathbf{K1}$-morphism $f^{\#}:B1 \to A1$ such that $\mathbf{F}(f^{\#});\varepsilon_{A2} = f$.

Then dually to Section 3.5.2 show how cofree objects induce *right adjoints*. Finally, prove facts dual to Propositions 3.5.19 and 3.5.20, thus proving that right adjoints and cofree objects give another equivalent definition of adjunction. □

Exercise 3.5.25. Develop yet another equivalent definition (at least for small categories) of an adjunction, centering around the bijection $\#:\mathbf{K1}(A1, \mathbf{G}(A2)) \to \mathbf{K2}(\mathbf{F}(A1), A2)$ using a generalised version of Hom-functors (cf. Example 3.4.15).

Proof sketch.

- For any small category $\mathbf{K}$ and two functors $\mathbf{F1}:\mathbf{K1} \to \mathbf{K}$ and $\mathbf{F2}:\mathbf{K2} \to \mathbf{K}$, define a functor $\mathbf{Hom}_{\mathbf{F1},\mathbf{F2}}:\mathbf{K1}^{op} \times \mathbf{K2} \to \mathbf{Set}$ by $\mathbf{Hom}_{\mathbf{F1},\mathbf{F2}}(\langle A1, A2 \rangle) = \mathbf{K}(\mathbf{F1}(A1), \mathbf{F2}(A2))$ and $\mathbf{Hom}_{\mathbf{F1},\mathbf{F2}}(\langle f1, f2 \rangle)(h) = \mathbf{F1}(f1);h;\mathbf{F2}(f2)$.
- Show that if $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ is left adjoint to $\mathbf{G}:\mathbf{K2} \to \mathbf{K1}$ then $\#:\mathbf{Hom}_{\mathbf{Id_{K1}},\mathbf{G}} \to \mathbf{Hom}_{\mathbf{F},\mathbf{Id_{K2}}}$ is a natural isomorphism.
- Finally, prove that for any functors $\mathbf{F}:\mathbf{K1} \to \mathbf{K2}$ and $\mathbf{G}:\mathbf{K2} \to \mathbf{K1}$, a natural isomorphism $\#:\mathbf{Hom}_{\mathbf{Id_{K1}},\mathbf{G}} \to \mathbf{Hom}_{\mathbf{F},\mathbf{Id_{K2}}}$ shows that $\mathbf{F}$ is left adjoint to $\mathbf{G}$. □

Exercise 3.5.26. Show that adjunctions compose: given any categories $\mathbf{K1}$, $\mathbf{K2}$ and $\mathbf{K3}$, and adjunctions $\langle \mathbf{F}, \mathbf{G}, \eta, \varepsilon \rangle$ from $\mathbf{K1}$ to $\mathbf{K2}$ and $\langle \mathbf{F}', \mathbf{G}', \eta', \varepsilon' \rangle$ from $\mathbf{K2}$ to $\mathbf{K3}$, we have an adjunction of the form $\langle \mathbf{F};\mathbf{F}', \mathbf{G}';\mathbf{G}, \_, \_ \rangle$ from $\mathbf{K1}$ to $\mathbf{K3}$. Fill in the holes! □

3.6 Bibliographical remarks

Category theory has found very many applications in computer science, and the material presented here covers just those fragments that we will require in later chapters. Books on category theory for mathematicians include the classic [Mac71] as well as the encyclopedic [HS73], with [AHS90] as a more recent favourite, the three-volume handbook [Bor94], the modestly-sized textbook [Awo06], and many more. An early book on category theory directed towards computer scientists is [AM75], followed by [Pie91], [Poi92] and [BW95]. An interesting angle is in [RB88], where categorical concepts are presented by coding them in ML.

Our terminology is mainly based on [Mac71], although we prefer to write composition in diagrammatic order, denoted by semicolon. The reader should be warned that the terminology and notation in category theory is not completely standardized, and differ from one author to another.

We have decided to keep to the basics, and have not ventured into many more advanced topics, some of which are quite important for computer-science applications. In particular, Cartesian closed categories [BW95], [Mit96] and the Curry-Howard isomorphism [SU06], categorical logic [LS86], monads [Man76], [Mog91], [Pho92], fibrations [Jac99], and topoi [Joh02], [Gol06] all deserve attention.

We have presented somewhat more material than usual on certain topics that will find application in some of the subsequent chapters. For example, in the material on factorisation systems (with Section 3.3 taken from [Tar85]) and on indexed categories (with Section 3.4.3.2 based on [TBG91]), we include some exercises which formulate facts that we will rely on later. We will work with indexed categories throughout the book, sometimes implicitly, since we find them more natural for these applications than equivalent formulations in terms of fibrations [Jac99].

We have deliberately chosen to use a notion of factorisation system based on [HS73]. The later book [AHS90] uses a somewhat more general concept, where factorisation morphisms are not required to be epi and mono, respectively, and therefore the uniqueness of the isomorphism between different factorisations of the same morphisms — or equivalently, of the diagonal in Lemma 3.3.4 — must be required explicitly. Although much of the material carries over, some results are simpler under our assumptions: for instance, we rely on Exercise 3.3.5 which does not hold in this form in the framework of [AHS90].

Our presentation of signatures, terms and algebras in Chapter 1 was elementary and set-theoretic, and we retain this style throughout the book. But category theory offers a whole spectrum of possibilities of doing universal algebra fruitfully in a different style. Exercises 3.4.26 and 3.4.41 relate to a categorical "Lawvere-style" presentation of some of the same concepts, see [Law63], [Man76], [BW85]. This was used in some early papers on algebraic specification, e.g. [GTWW75], but as it abstracts away from the choice of operation names in the signature, it seems less useful for applications to program specification. (This argument was put forward already in [BG80], with the notion of "signed theory" from [GB78] called to the rescue.) An alternative approach to specifications in this framework is given by sketches, see [BW95], which present specifications as graphs with indicated diagrams, cones and cocones that in a functorial model of the graph are mapped to commutative diagrams, limits and colimits, respectively. Commutative diagrams capture equational requirements here, with (co)limiting (co)cones offering additional specification power. Another related approach takes the general notion of a T-algebra for a functor $T : \mathbf{K} \to \mathbf{K}$ as its starting point, where a T-algebra on an object $A \in |\mathbf{K}|$ is a morphism from $T(A)$ to A; this works smoothly if T is a monad, see [Man76]. Such abstract approaches offer natural generalisations based on semantic interpretation in categories other than **Set**, but again, in our view, abstraction from familiar concepts and syntactic presentations makes them less convenient for practical use.

Chapter 4
Working within an arbitrary logical system

Several approaches to specification were discussed in Chapter 2. Each approach involved a different *logical system* as a part of its mathematical underpinnings. We encountered different definitions of:

Signatures: "ordinary" many-sorted signatures, signatures containing *bool*, *true* and *false* (for final and reachable semantics), error signatures, order-sorted signatures;

Algebras (on a signature Σ): "ordinary" Σ-algebras, error Σ-algebras, partial Σ-algebras, order-sorted Σ-algebras;

Logical sentences (on a signature Σ): Σ-equations, conditional Σ-equations, error Σ-equations (with safe and unsafe variables), Σ-definedness formulae, order-sorted Σ-equations; and

Satisfaction (of a Σ-sentence by a Σ-algebra): of a Σ-equation by a (total) Σ-algebra, of an error Σ-equation by an error Σ-algebra, of a Σ-equation by a partial Σ-algebra, of a Σ-definedness formula by a partial Σ-algebra, of an order-sorted Σ-equation by an order-sorted Σ-algebra.

All of these choices can be combined to obtain many different logical systems and hence different approaches to specification, e.g. partial error specifications with conditional axioms. Not only that, but there are several alternative approaches to the specification of partial algebras and at least half a dozen to the specification of error handling. Furthermore, there are many other variations that have not been considered, including the following (some of them briefly mentioned in Section 2.7.6):

- polymorphic signatures which permit polymorphic type constructors (rather than just sorts) and operations having polymorphic types;
- continuous algebras to handle infinite data objects such as streams;
- higher-order algebras to handle higher-order functions (i.e. functions taking functions as arguments and/or yielding functions as results);
- relational structures to model specifications containing predicates;
- inequations and conditional inequations;
- first-order formulae, with and without equality;

- various modal logics, including algorithmic, dynamic, and temporal logics, for formulating properties of (possibly non-functional) programs.

Some of these variations depart quite considerably from the usual algebraic framework presented in Chapters 1 and 2. But none of them (and very few of the others considered in the literature) are artificial, resulting merely from a theoretician's toying with formal definitions. All of them arise from the practical need to specify different aspects of software systems, often reflected by diverse features of different programming languages.

The resulting wealth of choice of definitions of the basic concepts is not a bad thing. None of the logical systems used in specifications is clearly better than all the others — and we should not expect that such a "best" system will ever be developed. In theory, we can imagine putting all of the above concepts together, producing a single logical system where signatures, algebras, sentences and the satisfaction relation would cover as special cases all we have considered up to now. But the result would be so huge and complex as to make it unmanageable. Moreover, what would we do if one day somebody points out that yet another view of software is important and should be reflected in specifications, and hence included in the logical system we use? Scrap everything and start again?

Different specification tasks may call for different systems to express most conveniently the properties required. Moreover, different logical systems may be appropriate for describing different aspects of the same software system, and so a number of logical systems may be useful in a single specification task. It is thus important that the designer of a software system be able to choose which logical system(s) to use.

An unfortunate effect of this necessary wealth of choice is that research on specification sometimes appears to be a confused mess, where everybody adopts a different combination of basic definitions. This makes it difficult to build on the work of others, to compare the results obtained for different logical systems, and to transfer results from one system to another. This is even more disturbing when one realises that such results include not only mathematical definitions and theorems, but also practically useful tools supporting software specification, development and verification produced at great expense of effort, time and money.

In fact, much of the work done turns out to be independent of the particular choice of the basic definitions, although this is often not obvious. The main objective of this chapter, and one of the main objectives of this book, is to lay out the mathematical foundations necessary to make this independence explicit. We achieve this using the notion of an *institution* which formalises the informal concept of a logical system devised to fit the purposes of specification theory; see Section 4.1 below for the definition. Our thesis is that building as much as possible on the notion of an institution brings important benefits for both the theory and the practice of software specification and development. On one hand, this allows much work on theories, results, and practical tools to be done just once for many different specific logical systems; on the other hand it forces, via abstraction, a better understanding of and deeper insight into the real problems.

A first example of this general approach is given in Section 4.2, where we recast the fundamental ideas of the standard approach to specification from Chapter 2 in the framework of an arbitrary institution.

It should be stressed that the notion of an institution captures only certain aspects of the informal concept of a logical system. In particular, it takes a model-theoretic view of logical systems, and no direct attempt is made to accommodate proof-theoretic concepts. See Section 9.1 for a discussion of how proof fits into the picture.

When discussing different approaches to specification in Chapter 2, apart from various basic notions of signature, algebra, sentence and satisfaction, we also considered different kinds of models (algebras satisfying a set of axioms) as particularly interesting:

- the initial models;
- the reachable models satisfying $\forall \varnothing \bullet \mathit{true} \neq \mathit{false}$;
- the final models in the category of reachable models satisfying $\forall \varnothing \bullet \mathit{true} \neq \mathit{false}$.

These options, although important for the overall style of specification, are of a different nature than the choice of the basic definitions embodied in the particular institution used. We show in Section 4.3 how such "interesting models" may be singled out in an arbitrary institution, thus suggesting that the choice here is in a sense orthogonal to the choice of the underlying institution.

Our general programme is to strive to work in an arbitrary institution as much as possible. However, the concepts involved in the basic theory of institutions are often too general, and hence too weak, to express all that is necessary. When this happens, it would be premature to give up, and switch to working in a particular institution. The "game" is then to identify a (hopefully) minimal set of additional assumptions under which the job can be done, covering most or all of the logical systems of interest. This gives rise to an enriched notion of institution with some additional structure that is relevant to the particular purpose we have in mind. A few examples of this are given in Sections 4.4 and 4.5.

Before proceeding we should warn the reader that although working in an arbitrary institution is very important, it is only one side of the story. The other side is to define an institution appropriate for the needs of the particular task at hand, and quite often this is far from trivial. Indeed, in many areas of Computer Science, the fundamental problem yet to be satisfactorily solved is the development of a logical system appropriate for the aspects of computing addressed. An example of an area for which a satisfactory, commonly accepted solution still seems to be outstanding (despite numerous proposals and active research) is the theory of concurrency.

4.1 Institutions

Following Goguen and Burstall [GB92], we introduce the notion of an *institution*, capturing some essential aspects of the informal concept of a "logical system". The

basic ingredients of an institution are: a notion of a signature in the system, and then for each signature, notions of an algebra with this signature, of a logical sentence over this signature, and finally a satisfaction relation between algebras and sentences.

In contrast to classical logic and model theory, we are not content with considering logical systems "pointwise", for an "arbitrary but fixed" signature. To capture the process of building a specification and designing a software system, some means of moving from one signature to another is required, that is, some notion of signature morphism. These typically enable signatures to be extended by new components, renaming and/or identifying others, as well as hiding some components used "internally" but not intended to be visible "externally". Any signature morphism should give rise to a translation of sentences and a translation of algebras determined by the change of names involved. Furthermore, these translations must be consistent with one another, preserving the satisfaction relation. As usual, when we switch from syntax (signatures, sentences) to semantics (algebras), the direction of translation is reversed.

The language of category theory is used in the definition to express the above ideas. This concisely and elegantly captures structure arising from signature morphisms, as well as forcing an appropriate level of generality and abstraction.

Definition 4.1.1 (Institution). An *institution* **INS** consists of:

- a category $\mathbf{Sign}_{\mathbf{INS}}$ of *signatures*;
- a functor $\mathbf{Sen}_{\mathbf{INS}}: \mathbf{Sign}_{\mathbf{INS}} \to \mathbf{Set}$, giving a set $\mathbf{Sen}(\Sigma)$ of Σ-*sentences* for each signature $\Sigma \in |\mathbf{Sign}_{\mathbf{INS}}|$ and a function $\mathbf{Sen}_{\mathbf{INS}}(\sigma): \mathbf{Sen}_{\mathbf{INS}}(\Sigma) \to \mathbf{Sen}_{\mathbf{INS}}(\Sigma')$ translating Σ-sentences to Σ'-sentences for each signature morphism $\sigma: \Sigma \to \Sigma'$;
- a functor $\mathbf{Mod}_{\mathbf{INS}}: \mathbf{Sign}_{\mathbf{INS}}^{op} \to \mathbf{Cat}$, giving a category $\mathbf{Mod}(\Sigma)$ of Σ-*models* for each signature $\Sigma \in |\mathbf{Sign}_{\mathbf{INS}}|$ and a functor $\mathbf{Mod}_{\mathbf{INS}}(\sigma): \mathbf{Mod}_{\mathbf{INS}}(\Sigma') \to \mathbf{Mod}_{\mathbf{INS}}(\Sigma)$ translating Σ'-models to Σ-models (and Σ'-morphisms to Σ-morphisms) for each signature morphism $\sigma: \Sigma \to \Sigma'$; and
- for each $\Sigma \in |\mathbf{Sign}_{\mathbf{INS}}|$, a *satisfaction relation* $\models_{\mathbf{INS},\Sigma} \subseteq |\mathbf{Mod}_{\mathbf{INS}}(\Sigma)| \times \mathbf{Sen}_{\mathbf{INS}}(\Sigma)$

such that for any signature morphism $\sigma: \Sigma \to \Sigma'$ the translations $\mathbf{Mod}_{\mathbf{INS}}(\sigma)$ of models and $\mathbf{Sen}_{\mathbf{INS}}(\sigma)$ of sentences preserve the satisfaction relation, that is, for any $\varphi \in \mathbf{Sen}_{\mathbf{INS}}(\Sigma)$ and $M' \in |\mathbf{Mod}_{\mathbf{INS}}(\Sigma')|$:

$$M' \models_{\mathbf{INS},\Sigma'} \mathbf{Sen}_{\mathbf{INS}}(\sigma)(\varphi) \quad \text{iff} \quad \mathbf{Mod}_{\mathbf{INS}}(\sigma)(M') \models_{\mathbf{INS},\Sigma} \varphi \qquad [\textit{Satisfaction condition}]$$

□

We will freely use standard terminology, and for example say that a Σ-model M *satisfies* a Σ-sentence φ, or that φ *holds* in M, whenever $M \models_{\mathbf{INS},\Sigma} \varphi$.

The term "model" (which we use following [GB92]) thereby becomes overloaded: it is used to refer both to objects in the category $\mathbf{Mod}_{\mathbf{INS}}(\Sigma)$ and to the algebras which satisfy a given set of axioms (we will soon extend the latter terminology to an arbitrary institution in Section 4.2, and then to an arbitrary structured

specification in Chapter 5). Hopefully, this will not lead to confusion as the context will always determine which of the two meanings is meant. If in doubt, we will use "a Σ-model" (where Σ is a signature) for the former, and "a model of Φ" (where Φ is a set of sentences) for the latter meaning of the word.

Notation.

- When there is no danger of confusion, we will omit the subscript **INS** when referring to the components of an institution **INS**. Similarly, the subscript Σ on the satisfaction relations will often be omitted.
- For any signature morphism $\sigma: \Sigma \to \Sigma'$, the function $\mathbf{Sen}(\sigma): \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$ will be denoted simply by $\sigma: \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$ and the functor $\mathbf{Mod}(\sigma): \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ by $\_|_\sigma: \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$. Thus for any Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$, $\sigma(\varphi) \in \mathbf{Sen}(\Sigma')$ is its *σ-translation* to a Σ'-sentence, and for any Σ'-model $M' \in |\mathbf{Mod}(\Sigma')|$, $\mathbf{M}'|_\sigma \in |\mathbf{Mod}(\Sigma)|$ is its *σ-reduct* to a Σ-model. We will also refer to M' as a *σ-expansion* of $M'|_\sigma$. Using this notation, the satisfaction condition of Definition 4.1.1 may be expressed as follows: $M' \models \sigma(\varphi) \iff M'|_\sigma \models \varphi$.
- For any signature Σ, the satisfaction relation extends naturally to sets of Σ-sentences and classes[1] of Σ-models. Namely, for any set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences and model $M \in |\mathbf{Mod}(\Sigma)|$, $M \models \Phi$ means $M \models \varphi$ for all $\varphi \in \Phi$. Then, for any Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$ and class $\mathscr{M} \subseteq |\mathbf{Mod}(\Sigma)|$ of Σ-models, $\mathscr{M} \models \varphi$ means $M \models \varphi$ for all $M \in \mathscr{M}$. Finally, we will also write $\mathscr{M} \models \Phi$ with the obvious meaning.
- For any signature Σ, we will sometimes write $Mod(\Sigma)$ for the class $|\mathbf{Mod}(\Sigma)|$ of all Σ-models. □

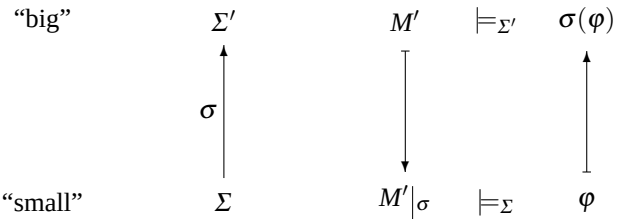
The definition of an institution as given above is very general and covers many logical systems of interest, as illustrated by the examples below. Nevertheless, it does impose some restrictions which should be made explicit before we proceed further.

First, the assumption that the translations of sentences and models induced by signature morphisms are functors may seem overly restrictive. In some situations it would be natural to relax the requirement of functoriality and assume that **Sen** (and perhaps **Mod** as well) is a functor only "up to some appropriate equivalence". For example, given two signature morphisms $\sigma: \Sigma \to \Sigma'$ and $\sigma': \Sigma' \to \Sigma''$, for any sentence $\varphi \in \mathbf{Sen}(\Sigma)$ it follows from the functoriality of **Sen** that $\mathbf{Sen}(\sigma;\sigma')(\varphi) = \mathbf{Sen}(\sigma')(\mathbf{Sen}(\sigma)(\varphi))$ (or using the notational convention introduced above, $(\sigma;\sigma')(\varphi) = \sigma'(\sigma(\varphi))$). This seems overly restrictive when, for example, local identifiers or bound variables are used in sentences. All we really care about here is that the two translations of φ to a Σ''-sentence are *semantically equivalent*: that $(\sigma;\sigma')(\varphi)$ and $\sigma'(\sigma(\varphi))$ hold in the same Σ''-models. A solution

[1] We will be somewhat more careful about the set-theoretical foundations than in our presentation of the basics of category theory in Chapter 3: we will refer to collections of sentences as "sets" and to collections of models as "classes", as in Chapter 2. This is consistent with the formal definition of an institution above, and satisfactory for the logical systems formalised as institutions given as examples (but see Example 4.1.46, footnote 16).

is to consider sentences up to this semantic equivalence, and work in an institution where sentences simply *are* the corresponding equivalence classes. This solution would resemble the usual practice in λ-calculi, where terms are considered "up to α-conversion" (renaming of bound variables), meaning that terms are really classes of mutually α-convertible syntactic terms.

The only explicit requirement in the definition of an institution is that the satisfaction condition holds. Speaking informally, this deals with the situation where a "small" signature Σ and a "big" signature Σ' are related by a signature morphism $\sigma:\Sigma \to \Sigma'$, and we have a model $M' \in |\mathbf{Mod}(\Sigma')|$ over the "big" signature, and a sentence $\varphi \in \mathbf{Sen}(\Sigma)$ over the "small" signature. There are then two ways to check whether M' "satisfies" φ: we can either reduce the model M' to the "small" signature and check whether the reduct satisfies the sentence φ, or translate the sentence φ to the "big" signature and check whether the translated sentence holds in the model M'.



The satisfaction condition states that these two alternatives are equivalent. This embodies two fundamental assumptions. One is that the meaning of a sentence depends only on the components used in the sentence, and does not depend on the context in which the sentence is considered. The other is that the meaning of a sentence is preserved under translation; as [GB92] say:

Truth is invariant under change of notation.

The latter requirement does not raise much doubt — we are not aware of any natural system in which it would not hold. The former, however, is sometimes violated. There are natural logical systems where the meaning of a sentence depends on the context in which it is used, or in other words on the signature over which the sentence is considered. For instance, in logical systems involving quantifiers, the range of quantification may implicitly depend on the signature, with quantified variables ranging only over reachable values, so that "$\exists x \bullet \ldots$" is interpreted as "there exists an element x which is the value of a ground term, such that $\ldots$" and similarly for universal quantification. For such a logic the satisfaction condition does not hold unless very strong restrictions are placed on signature morphisms.

Exercise 4.1.2. Give a concrete counterexample to the satisfaction condition for a logical system similar to equational logic, but with the universally quantified variables in equations ranging only over reachable values. Show how the logical system you give may be modified to make the satisfaction condition hold. HINT: The satisfaction condition failed because the interpretation of universal quantification over

reachable values implicitly depends on the signature; try to make this dependence explicit! □

4.1.1 Examples of institutions

Example 4.1.3 (Ground equational logic GEQ). The institution **GEQ** of ground equational logic is defined as follows:

- The category $\mathbf{Sign}_{\mathbf{GEQ}}$ is just **AlgSig**, the usual category of algebraic signatures.
- The functor $\mathbf{Sen}_{\mathbf{GEQ}}\colon \mathbf{AlgSig} \to \mathbf{Set}$ gives:
  - the set of ground Σ-equations for each $\Sigma \in |\mathbf{AlgSig}|$; and
  - the σ-translation function taking ground Σ-equations to ground Σ'-equations for each signature morphism $\sigma\colon \Sigma \to \Sigma'$.
- The functor $\mathbf{Mod}_{\mathbf{GEQ}}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ is the functor $\mathbf{Alg}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ as defined in Example 3.4.29, that is, $\mathbf{Mod}_{\mathbf{GEQ}}$ gives:
  - the category $\mathbf{Alg}(\Sigma)$ of Σ-algebras and Σ-homomorphisms for each $\Sigma \in |\mathbf{AlgSig}|$; and
  - the reduct functor $\_|_\sigma\colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$ mapping Σ'-algebras and Σ'-homomorphisms to Σ-algebras and Σ-homomorphisms for each signature morphism $\sigma\colon \Sigma \to \Sigma'$.
- For each $\Sigma \in |\mathbf{AlgSig}|$, the satisfaction relation $\models_{\mathbf{GEQ},\Sigma} \subseteq |\mathbf{Alg}(\Sigma)| \times \mathbf{Sen}_{\mathbf{GEQ}}(\Sigma)$ is the usual relation of satisfaction of a ground Σ-equation by a Σ-algebra.

The Satisfaction Lemma (Lemma 2.1.8) ensures that the required satisfaction condition holds and so that the above definition indeed yields an institution. □

Example 4.1.4 (Equational logic EQ). The institution **EQ** of (ordinary) equational logic is defined as follows:

- The category $\mathbf{Sign}_{\mathbf{EQ}}$ is just **AlgSig**.
- The functor $\mathbf{Sen}_{\mathbf{EQ}}\colon \mathbf{AlgSig} \to \mathbf{Set}$ gives:
  - the set of Σ-equations for each $\Sigma \in |\mathbf{AlgSig}|$; and
  - the σ-translation function taking Σ-equations to Σ'-equations for each signature morphism $\sigma\colon \Sigma \to \Sigma'$.[2]
- The functor $\mathbf{Mod}_{\mathbf{EQ}}$ is $\mathbf{Alg}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$, just like $\mathbf{Mod}_{\mathbf{GEQ}}$ for ground equational logic.

[2] The exact treatment of variables in equations requires special care to ensure that the translation of equations along possibly non-injective signature morphisms is indeed functorial. The use of disjoint union in the translation of many-sorted sets of variables in Definition 1.5.10 causes problems here. The simplest way to make this work is to assume that, in each equation, variables of different sorts are distinct. See [GB92] for details.

- For each $\Sigma \in |\mathbf{AlgSig}|$, the satisfaction relation $\models_{\mathbf{EQ},\Sigma} \subseteq |\mathbf{Alg}(\Sigma)| \times \mathbf{Sen_{EQ}}(\Sigma)$ is the usual relation of satisfaction of a Σ-equation by a Σ-algebra.

The Satisfaction Lemma (Lemma 2.1.8) again ensures that the required satisfaction condition holds and so that the above definition indeed yields an institution. □

There is an obvious sense in which **GEQ** can be regarded as a "subinstitution" of **EQ**. We will encounter further such cases below. We refrain from formulating a notion of subinstitution because the concept turns out to be more subtle than it might appear at first. We postpone a proper treatment of relationships between institutions to Chapter 10 (in particular, see Exercise 10.4.8).

Exercise 4.1.5 (Reachable ground equational logic RGEQ). Define an institution **RGEQ** of ground equational logic on reachable algebras, by modifying the definion of **GEQ** so that only reachable algebras are considered as models. Do not forget to adjust the definition of reduct functors!

Try to extend this to an institution **REQ** of equational logic on reachable algebras — and notice that the satisfaction condition cannot be ensured without modifying the notion of an equation to include "data constructors" to determine the reachable values for which the equation is to be considered, as already hinted at in Exercise 4.1.2. □

Example 4.1.6 (Partial equational logic PEQ). The institution **PEQ** of partial equational logic is defined as follows (cf. Section 2.7.4):

- $\mathbf{Sign_{PEQ}}$ is **AlgSig** again.
- $\mathbf{Sen_{PEQ}}: \mathbf{AlgSig} \to \mathbf{Set}$ gives:
  - the set of Σ-equations and Σ-definedness formulae for each $\Sigma \in |\mathbf{AlgSig}|$; and
  - the σ-translation function taking Σ-equations and Σ-definedness formulae to Σ'-equations and Σ'-definedness formulae for each signature morphism $\sigma: \Sigma \to \Sigma'$.[3]
- $\mathbf{Mod_{PEQ}}: \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ gives:
  - the category $\mathbf{PAlg}(\Sigma)$ of partial Σ-algebras and weak Σ-homomorphisms for each $\Sigma \in |\mathbf{AlgSig}|$ (cf. Example 3.3.13); and
  - the reduct functor $\_|_\sigma: \mathbf{PAlg}(\Sigma') \to \mathbf{PAlg}(\Sigma)$ defined similarly as in the total case for each signature morphism $\sigma: \Sigma \to \Sigma'$.
- For each $\Sigma \in |\mathbf{AlgSig}|$, the satisfaction relation $\models_{\mathbf{PEQ},\Sigma} \subseteq |\mathbf{PAlg}(\Sigma)| \times \mathbf{Sen_{PEQ}}(\Sigma)$ is the satisfaction of Σ-equations (with strong equality) and Σ-definedness formulae by partial Σ-algebras.

Exercise. Proceeding similarly as in the proof of Satisfaction Lemma (Lemma 2.1.8), show that the satisfaction condition holds for **PEQ**. □

[3] As in Example 4.1.4, care is needed with the treatment of variables and their translation under signature morphisms, see footnote 2.

Example 4.1.7 (Ground partial equational logic PGEQ). The institution **PGEQ** of ground partial equational logic is defined just like the institution **PEQ** of partial equational logic above, except that only ground equations and ground definedness formulae are considered. □

Exercise 4.1.8. Recalling the notion of existential equality for partial algebras from Section 2.7.4, define institutions $\mathbf{PEQ}^e$ and $\mathbf{PGEQ}^e$ of partial existence equational logic and ground partial existence equational logic, respectively, modifying the definitions in Examples 4.1.6 and 4.1.7 by using existential equations of the form $\forall X.t \stackrel{e}{=} t'$ and their ground versions only. □

Example 4.1.9 (Propositional logic PROP). The institution **PROP** of propositional logic is defined as follows:

- $\mathbf{Sign}_{\mathbf{PROP}}$ is **Set**, the usual category of sets. In this context, for each "signature" $P \in |\mathbf{Set}|$, we call elements of P *propositional variables*.
- $\mathbf{Sen}_{\mathbf{PROP}}: \mathbf{Set} \to \mathbf{Set}$ gives
  - For each $P \in |\mathbf{Set}|$, $\mathbf{Sen}_{\mathbf{PROP}}(P)$ is the least set that contains P, sentences true and false, and is closed under the usual propositional connectives, that is, if $\varphi, \varphi' \in \mathbf{Sen}_{\mathbf{PROP}}(\Sigma)$ then also $\varphi \vee \varphi' \in \mathbf{Sen}_{\mathbf{PROP}}(\Sigma)$, $\neg\varphi \in \mathbf{Sen}_{\mathbf{PROP}}(\Sigma)$, $\varphi \wedge \varphi' \in \mathbf{Sen}_{\mathbf{PROP}}(\Sigma)$, and $\varphi \Rightarrow \varphi' \in \mathbf{Sen}_{\mathbf{PROP}}(\Sigma)$. [4]
  - For each function $\sigma: P \to P'$, $\mathbf{Sen}_{\mathbf{PROP}}(\sigma)$ extends σ to the translation of arbitrary propositional sentences with propositional variables in P to propositional sentences with propositional variables in P', preserving the propositional connectives in the obvious way.
- $\mathbf{Mod}_{\mathbf{PROP}}: \mathbf{Set}^{op} \to \mathbf{Cat}$ gives:
  - For each set of propositional variables $P \in |Set|$, P-models are all functions from P to $\{ff, tt\}$. These functions can be identified with subsets of P, where $M: P \to \{ff, tt\}$ yields $\{p \in P \mid M(p) = tt\}$). Model morphisms are just inclusions of these sets, i.e., given two P-models $M_1, M_2: P \to \{ff, tt\}$, we have a (unique) morphism from M_1 to M_2 if for all $p \in P$, $M_2(p) = tt$ whenever $M_1(p) = tt$.
  - For each signature morphism $\sigma: P \to P'$, the reduct functor $\mathbf{Mod}_{\mathbf{PROP}}(\sigma): \mathbf{Mod}_{\mathbf{PROP}}(P') \to \mathbf{Mod}_{\mathbf{PROP}}(P)$ maps any model $M': P' \to \{ff, tt\}$ to $\sigma; M': P \to \{ff, tt\}$.
- For each $P \in |\mathbf{Set}|$, the satisfaction relation $\models_{\mathbf{PROP},P} \subseteq |\mathbf{Mod}_{\mathbf{PROP}}(P)| \times \mathbf{Sen}_{\mathbf{PROP}}(P)$ is the usual relation of satisfaction of propositional sentences, that is, for any P-model $M: P \to \{ff, tt\}$, $p \in P$ and $\varphi, \varphi' \in \mathbf{Sen}_{\mathbf{PROP}}(P)$:
  - $M \models_{\mathbf{PROP},P} p$ if and only if $M(p) = tt$,
  - $M \models_{\mathbf{PROP},P} \varphi \vee \varphi'$ if and only if $M \models_{\mathbf{PROP},P} \varphi$ or $M \models_{\mathbf{PROP},P} \varphi'$,
  - $M \models_{\mathbf{PROP},P} \neg\varphi$ if and only if $M \not\models_{\mathbf{PROP},P} \varphi$,
  - $M \models_{\mathbf{PROP},P} \varphi \wedge \varphi'$ if and only if $M \models_{\mathbf{PROP},P} \varphi$ and $M \models_{\mathbf{PROP},P} \varphi'$.

[4] We tacitly assume here that true, false, $\vee$, $\wedge$, $\Rightarrow$, $\neg$ are new symbols (not in P), and rely on the usual precedence rules and parentheses to make sure that no ambiguities in their "parsing" arise.

- $M \models_{\mathbf{PROP},P} \varphi \Rightarrow \varphi'$ if and only if $M \models_{\mathbf{PROP},P} \varphi'$ or $M \not\models_{\mathbf{PROP},P} \varphi$
- $M \models_{\mathbf{PROP},P} \mathsf{true}$, and
- $M \not\models_{\mathbf{PROP},P} \mathsf{false}$. □

Exercise 4.1.10. Recall the specification of Boolean algebras in Example 2.2.4.

Note that one way to view the definitions in Example 4.1.9 is to define the set of P-sentences as Boolean terms with variables from P. Then, one can consider the two-element Boolean algebra $\mathbb{B}$ with the carrier $\{ff, tt\}$ (with $\mathsf{true}_{\mathbb{B}} = tt$ and $\mathsf{false}_{\mathbb{B}} = ff$). Furthermore, any propositional model $M : P \to \{ff, tt\}$ induces evaluation of terms $M^\sharp : \mathbf{Sen}_{\mathbf{PROP}}(P) \to |\mathbb{B}|$, with $M^\sharp(\varphi) = tt$ if and only if $M \models_{\mathbf{PROP},P} \varphi$ as defined above.

Define another institution of propositional logic, $\mathbf{PROP}^{\mathbf{BA}}$, where signatures and sentences are as in **PROP**, but models use arbitrary Boolean algebras rather than just $\mathbb{B}$. That is, for any set $P \in |\mathbf{Set}|$ of propositional variables, a P-model in $\mathbf{PROP}^{\mathbf{BA}}$ consists of a Boolean algebra B together with valuation $M : P \to |B|$, where we define $\langle B, M\rangle \models_{\mathbf{PROP}^{\mathbf{BA}},P} \varphi$ if and only if $\varphi_B(M) = \mathsf{true}_B$ (where $\varphi_B(M)$ is the value of term φ in B under valuation M).

Prove now that the semantic consequence relation (Definition 2.3.6, cf. Definition 4.2.5 below) in **PROP** and $\mathbf{PROP}^{\mathbf{BA}}$ coincide.

HINT: Clearly, if $\Psi \models_{\mathbf{PROP}^{\mathbf{BA}},P} \varphi$ then also $\Psi \models_{\mathbf{PROP},P} \varphi$ for any set P of propositional variables, $\Psi \subseteq \mathbf{Sen}_{\mathbf{PROP}}(P)$ and $\varphi \in \mathbf{Sen}_{\mathbf{PROP}}(P)$. Suppose now that $\Psi \not\models_{\mathbf{PROP}^{\mathbf{BA}},P} \varphi$. Use the following lemma[5]:

Lemma. *Given any Boolean algebra B and element $b \in |B|$ such that $b \neq \mathsf{true}_B$, there exists a homomorphism $h : B \to \mathbb{B}$ from B to the two-element Boolean algebra $\mathbb{B}$ such that $h(b) = \mathsf{false}_{\mathbb{B}}$.*

Now, given any Boolean algebra B and valuation $M : P \to |B|$ such that for all $\psi \in \Psi$, $\psi_B(M) = \mathsf{true}_B$ and $\varphi_B(M) \neq \mathsf{true}_B$, conclude using the above lemma that $(M;h)^\sharp(\psi) = tt$ for all $\psi \in \Psi$, while $(M;h)^\sharp(\varphi) = ff$. □

Exercise 4.1.11. Define the institution of intuitionistic propositional logic, $\mathbf{PROP}^{\mathbf{I}}$, following the pattern of $\mathbf{PROP}^{\mathbf{BA}}$ in Exercise 4.1.10, but using arbitrary Heyting algebras (see Example 2.7.6) rather than just Boolean algebras.

Show that if $\Psi \models_{\mathbf{PROP}^{\mathbf{I}},P} \varphi$ then also $\Psi \models_{\mathbf{PROP},P} \varphi$ for any set P of propositional variables, $\Psi \subseteq \mathbf{Sen}_{\mathbf{PROP}}(P)$ and $\varphi \in \mathbf{Sen}_{\mathbf{PROP}}(P)$, and give a counterexample to show that the opposite implication fails in general. □

Example 4.1.12 (First-order predicate logic with equality FOPEQ). The institution **FOPEQ** of first-order predicate logic with equality is defined as follows:

- $\mathbf{Sign}_{\mathbf{FOPEQ}}$, from now on denoted by **FOSig**, is the category of *first-order signatures* where we define:

[5] The proof of this lemma is beyond the scope of this book, but see e.g. [RS63], I,8.5 and II,5.2,(a)⇒(e).

  - A *first-order signature* Θ is a triple $\langle S, \Omega, \Pi \rangle$, where S is a set (of *sort names*), $\Omega = \langle \Omega_{w,s} \rangle_{w \in S^*, s \in S}$ is a family of sets (of *operation names* with their arities and result sorts indicated — just as in algebraic signatures) and $\Pi = \langle \Pi_w \rangle_{w \in S^*}$ is a family of sets (of *predicate* or *relation names* with their arities indicated).
  - A *first-order signature morphism* $\theta : \langle S, \Omega, \Pi \rangle \to \langle S', \Omega', \Pi' \rangle$ consists again of three components: a function $\theta_{sorts} : S \to S'$, an $S^* \times S$-indexed family of functions $\theta_{ops} = \langle (\theta_{ops})_{w,s} : \Omega_{w,s} \to \Omega'_{\theta^*_{sorts}(w), \theta_{sorts}(s)} \rangle_{w \in S^*, s \in S}$ (these are as in algebraic signature morphisms) and $\theta_{preds} = \langle (\theta_{preds})_w : \Pi_w \to \Pi'_{\theta^*_{sorts}(w)} \rangle_{w \in S^*}$. (As with algebraic signature morphisms, all the components of a first-order signature morphism θ will be denoted by θ when there is no danger of ambiguity.)

- $\mathbf{Sen_{FOPEQ}} : \mathbf{FOSig} \to \mathbf{Set}$ gives:
  - For each first-order signature $\Theta = \langle S, \Omega, \Pi \rangle$, $\mathbf{Sen_{FOPEQ}}(\Theta)$ is the set of all closed (i.e. without unbound occurrences of variables) *first-order formulae* built out of atomic formulae using the standard propositional connectives ($\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$, $\neg$) and quantifiers ($\forall$, $\exists$). The *atomic formulae* are equalities of the form $t = t'$, where t and t' are $\langle S, \Omega \rangle$-terms (possibly with variables) of the same sort, atomic predicate formulae of the form $p(t_1, \ldots, t_n)$, where $p \in \Pi_{s_1 \ldots s_n}$ and t_1, …, t_n are terms (possibly with variables) of sorts s_1, …, s_n, respectively, and the logical constants true and false.
  - For each first-order signature morphism $\theta : \Theta \to \Theta'$, $\mathbf{Sen_{FOPEQ}}(\theta)$ is the translation of first-order Θ-sentences to first-order Θ'-sentences determined in the obvious way by the renaming θ of sort, operation and predicate names in Θ to the corresponding names in Θ'.[6]

- $\mathbf{Mod_{FOPEQ}} : \mathbf{FOSig}^{op} \to \mathbf{Cat}$, from now on denoted by **FOStr**, gives:
  - For each first-order signature $\Theta = \langle S, \Omega, \Pi \rangle$, the category $\mathbf{FOStr}(\Theta)$ of *first-order Θ-structures* is defined as follows:
    - A *first-order Θ-structure* $A \in |\mathbf{FOStr}(\Theta)|$ consists of a carrier set $|A|_s$ for each sort name $s \in S$, a function $f_A : |A|_{s_1} \times \ldots \times |A|_{s_n} \to |A|_s$ for each operation name $f \in \Omega_{s_1 \ldots s_n, s}$ (these are the same as in $\langle S, \Omega \rangle$-algebras) and a relation $p_A \subseteq |A|_{s_1} \times \ldots \times |A|_{s_n}$ for each predicate name $p \in \Pi_{s_1 \ldots s_n}$. In the following we write $p_A(a_1, \ldots, a_n)$ for $\langle a_1, \ldots, a_n \rangle \in p_A$.
    - For any first-order Θ-structures A and B, a *first-order Θ-morphism* between them, $h : A \to B$, is a family of functions $h = \langle h_s : |A|_s \to |B|_s \rangle_{s \in S}$ which preserves the operations (as ordinary $\langle S, \Omega \rangle$-homomorphisms do) and predicates (i.e., for $p \in \Pi_{s_1 \ldots s_n}$ and $a_1 \in |A|_{s_1}$, …, $a_n \in |A|_{s_n}$, if $p_A(a_1, \ldots, a_n)$ then $p_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$ as well). A Θ-morphism is *strong* if it reflects predicates as well, so that for $p \in \Pi_{s_1 \ldots s_n}$ and $a_1 \in |A|_{s_1}$, …, $a_n \in |A|_{s_n}$, $p_A(a_1, \ldots, a_n)$ if and only if $p_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$.

[6] As in Example 4.1.4, some care is needed with the exact treatment of quantified variables and their translation under signature morphisms (cf. footnote 2) — again, the simplest solution is to assume that, in each formula, variables of different sorts are distinct. See [GB92] for a careful presentation.

  - For each first-order signature morphism $\theta:\Theta \to \Theta'$, we have the *$\theta$-reduct functor* $\mathbf{FOStr}(\theta):\mathbf{FOStr}(\Theta') \to \mathbf{FOStr}(\Theta)$ defined similarly as reduct functors corresponding to algebraic signature morphisms.

- For each $\Theta \in |\mathbf{FOSig}|$, the satisfaction relation $\models_{\mathbf{FOPEQ},\Theta} \subseteq |\mathbf{FOStr}(\Theta)| \times \mathbf{Sen}_{\mathbf{FOPEQ}}(\Theta)$ s the usual relation of satisfaction of first-order sentences in first-order structures, determined by the usual interpretation of $\vee$, $\wedge$, $\Rightarrow$ and $\neg$ as disjunction, conjunction, implication and negation, respectively, of $\forall$ and $\exists$ as universal and existential quantifiers, respectively, of equalities $t = t'$ as identity of the values of t and t', of atomic predicate formulae $p(t_1,\ldots,t_n)$ as the value of the predicate named p in the structure on the values of the terms t_1, ..., t_n, and of true and false.

Exercise. Work out all the details omitted from the above definition. Then, generalising the proof of the Satisfaction Lemma, show that the satisfaction condition holds for **FOPEQ**. □

Exercise 4.1.13 (First-order predicate logic FOP, first-order logic with equality FOEQ). First-order predicate logic with equality contains some standard "sublogics". Define the institution **FOP** of first-order predicate logic (without equality), by referring to the same signatures and models as in **FOPEQ**, but limiting the sentences to those that do not contain equality.

Define also the institution **FOEQ** with signatures and models as in the institution **EQ** of equational logic, but with first-order sentences (without predicates). □

Exercise 4.1.14 (Infinitary logics). Define an institution of so-called $L_{\omega_1\omega}$ logic, which extends first-order predicate logic with equality by allowing conjunctions and disjunctions of *countable* families of formulae (but still only finitary quantification). Extend this further by allowing quantification over countable sets of variables, obtaining an institution of $L_{\omega_1\omega_1}$ logic. You may also want to define institutions of $L_{\alpha\beta}$ logics, for any infinite cardinal numbers α and β such that $\beta \leq \alpha$, with conjunctions and disjunctions of sets of formulae of cardinality smaller than α and quantification over sets of variables of cardinality smaller than β. □

Exercise 4.1.15 (Higher-order logics). Define an institution of *second-order logic*, which extends first-order logic by introducing variables ranging over predicates (which in a model denote subsets of a product of the carrier sets) and quantification over such (first-order) predicates. Then generalise this further to an institution of *higher-order logic*, which introduces variables that range over (second-order) predicates with arities that may include arities of first-order predicates, and predicates with arities that may include arities of second-order predicates, etc., and allows for quantification over such higher-order predicates. Without much additional effort, you may want to extend this further, by allowing variables that range over functions of an arbitrary higher-order type, and quantification over such functions. Note though that this will be different from first-order logic for higher-order algebras as sketched in Example 2.7.56, where quantification over higher-order function types does not necessarily coincide with quantification over *all* functions of this type. □

Exercise 4.1.16 (First-order equational logic with boolean values FOEQBool). Define an institution **FOEQBool** which differs from **FOEQ** by considering only signatures that contain a subsignature Σ_{bool} of *truth values* (Σ_{bool} has a special, distinguished sort *bool* and two constants $true, false : bool$) and assuming that signature morphisms preserve and reflect symbols in Σ_{bool} and that algebras interpret them in the standard way (the carrier of sort *bool* has exactly two distinct elements that are values of *true* and *false*, respectively).

There is now an obvious equivalence between the categories of signatures of **FOPEQ** and **FOEQBool** obtained by mapping each first-order signature to the algebraic signature with the sort *bool* and constants $true, false : bool$ added, and with new operation name $f_p : s_1 \times \ldots \times s_n \to bool$ for each predicate $p : s_1 \times \ldots \times s_n$. First-order structures give raise to algebras with the standard interpretation of Σ_{bool} and with functions f_p that yield the value of *true* exactly on those arguments for which the predicate p holds. Clearly, this yields a one-to-one correspondence between first-order structures and algebras over the corresponding signatures. However, this does not extend to model morphisms in general. (**Exercise:** Find a counterexample. Notice though that every *strong* morphism between first-order structures extends to a homomorphism between their corresponding algebras.) We then consider translation of atomic sentences $p(t_1, \ldots, t_n)$ to equalities $p(t_1, \ldots, t_n) = true$, and extend it further to arbitrary first-order sentences with predicates and equality in the obvious way.

Prove that such translations of sentences and models preserve and reflect satisfaction. □

It is not much more difficult to define, for example, the institution **PFOPEQ** of partial first-order predicate logic with equality, or any other institution formalising one of the many standard variants of the classical notions.

Exercise 4.1.17 (Partial first-order predicate logic with equality PFOPEQ). Define the institution **PFOPEQ** of partial first-order predicate logic with equality according to the following sketch:

- $\mathbf{Sign}_{\mathbf{PFOPEQ}} = \mathbf{FOSig}$.
- For each $\Theta \in |\mathbf{FOSig}|$, partial first-order Θ-sentences are defined in the same way as usual first-order Θ-sentences on atomic formulae which here include *atomic definedness formulae* $def(t)$ for any Θ-term t, in addition to equalities and atomic predicate formulae. The translation of sentences along signature morphisms is defined in the obvious way.
- For each $\Theta \in |\mathbf{FOSig}|$, the models in $\mathbf{Mod}_{\mathbf{PFOPEQ}}(\Theta)$ are like first-order Θ-structures except that the operations may be partial. Morphisms in $\mathbf{Mod}_{\mathbf{PFOPEQ}}(\Theta)$ are like first-order Θ-morphisms but are required to preserve definedness of operations, as weak homomorphisms of partial algebras do. The reduct functors are defined similarly as for first-order structures.
- For each signature $\Theta \in |\mathbf{FOSig}|$, the satisfaction relation $\models_{\mathbf{PFOPEQ},\Theta}$ is defined like the usual first-order satisfaction relation, building on the interpretation of atomic equalities and definedness formulae which follows the interpretation of

(strong) equations and definedness formulae in partial algebras as defined in the institution **PEQ** of partial equational logic and on the usual interpretation of atomic predicate formulae $p(t_1,\ldots,t_n)$ which yields *false* when any of t_1, ..., t_n is undefined. □

Exercise 4.1.18 (Partial first-order logic with equality PFOEQ). Following Exercise 4.1.13, define the institution **PFOEQ** of partial first-order logic with equality with signatures and models inherited from the institution **PEQ** of partial equational logic, but with first-order sentences (without predicates). Similarly, define the institution **PFOP** of partial first-order predicate logic (without equality). □

Exercise 4.1.19 (Partial first-order equational logic with truth PFOEQTruth). As in Exercise 4.1.16, define now an institution **PFOEQBool** of partial first-order logic with equality and built-in boolean values.

However, using partial functions predicates may be modelled differently (and more faithfully when model morphisms are considered). Define an institution **PFOEQTruth** which differs from **PFOEQ** by assuming that the signatures contain a subsignature Σ_{truth} (which has a special, distinguished sort *truth* with a single constant *true*:*truth*), that signature morphisms preserve and reflect symbols in Σ_{truth}, and that algebras interpret them in the standard way: the carrier of sort *truth* has exactly one element that is the value of *true*.

The equivalence of categories of signatures and the translation of sentences between **PFOPEQ** and **PFOEQTruth** can now be given in essentially the same way as in Exercise 4.1.16. Moreover, first-order partial structures are in one-to-one correspondence with algebras over the corresponding algebraic signature, and this correspondence may be described exactly as in Exercise 4.1.16 as well. The difference is that now for arguments for which predicates do not hold, their corresponding operations are undefined instead of yielding a non-*true* value. This allows us to extend this correspondence to model morphisms as well.

Prove that such translations of sentences and models preserve and reflect satisfaction. □

Exercise 4.1.20. Recall the notion of a strong homomorphism between partial algebras (Definition 2.7.31) and between first-order structures (given in Example 4.1.12). For each of the institutions above with models that involve partial operations or predicates (**FOPEQ**, **FOP**, **PFOPEQ**, **PEQ**, etc.) define a variant in which all morphisms are strong. We will refer to these institutions as $\mathbf{FOPEQ_{str}}$, $\mathbf{FOP_{str}}$, $\mathbf{PFOPEQ_{str}}$, $\mathbf{PEQ_{str}}$, etc. In particular, model morphisms in $\mathbf{PFOPEQ_{str}}$ preserve and reflect predicates as well as definedness of operations. □

Exercise 4.1.21. Using the material in Sections 2.7.1, 2.7.3 and 2.7.5, respectively, define institutions: $\mathbf{EQ}^{\Rightarrow}$ of conditional equations with signatures and models as in **EQ**; **Horn** of Horn formulae built over signatures and models of **FOPEQ**, where sentences have the form $\forall X \bullet \varphi_1 \wedge \ldots \wedge \varphi_n \Rightarrow \varphi$ for atomic formulae φ_1, ..., φ_n, φ; **ErrEQ** of error equational logic; and **OrdEQ** of order-sorted equational logic; □

Example 4.1.22 (The institution CEQ of equational logic for continuous algebras). We need some auxiliary definitions. Let $\Sigma = \langle S, \Omega \rangle$ be an algebraic signature.

Recall (cf. Example 3.3.14) that a continuous Σ-algebra $A \in |\mathbf{CAlg}(\Sigma)|$ consists of carriers, which are complete partial orders $\langle |A|_s, \leq_s \rangle$ for $s \in S$, and operations, which are continuous functions $f_A : |A|_{s_1} \times \ldots \times |A|_{s_n} \to |A|_s$ for $f : s_1 \times \ldots \times s_n \to s$ in Σ.

For any S-sorted set X (of variables), the (S-sorted) set $|T^\infty_\Sigma(X)|$ of *infinitary Σ-terms* is the least set such that[7]:

- $X \subseteq |T^\infty_\Sigma(X)|$;
- for each $f : s_1 \times \ldots \times s_n \to s$ in Σ, if $t_1 \in |T^\infty_\Sigma(X)|_{s_1}$, ..., $t_n \in |T^\infty_\Sigma(X)|_{s_n}$ then $f(t_1, \ldots, t_n) \in |T^\infty_\Sigma(X)|_s$; and
- for each $s \in S$, if for $k \geq 0$, $t_k \in T^\infty_\Sigma(X)_s$, then $\bigsqcup \langle t_k \rangle_{k \geq 0} \in |T^\infty_\Sigma(X)|_s$.

Intuitively, $|T^\infty_\Sigma(X)|$ contains all the usual finitary Σ-terms and in addition is closed under formal "least upper bounds" of countable sequences of terms. Notice, however, that we do not provide $|T^\infty_\Sigma(X)|$ with the structure of a continuous Σ-algebra; in particular, a term $\bigsqcup \langle t_k \rangle_{k \geq 0}$ is just a formal expression here, not a least upper bound.

Then, for any continuous Σ-algebra A and valuation of variables $v : X \to |A|$, we define a *partial* function $v^\# : |T^\infty_\Sigma(X)| \to |A|$ which for any term $t \in |T^\infty_\Sigma(X)|$ yields the *value* $v^\#(t)$ *of* t (if defined):

- for $x \in X$, $v^\#(x) = v(x)$;
- for $f : s_1 \times \ldots \times s_n \to s$ and $t_1 \in |T^\infty_\Sigma(X)|_{s_1}$, ..., $t_n \in |T^\infty_\Sigma(X)|_{s_n}$, $v^\#(f(t_1, \ldots, t_n))$ is defined if and only if $v^\#(t_1)$, ..., $v^\#(t_n)$ are all defined, and then $v^\#(f(t_1, \ldots, t_n)) = f_A(v^\#(t_1), \ldots, v^\#(t_n))$; and
- for $t_k \in T^\infty_\Sigma(X)_s$, $k \geq 0$, $v^\#(\bigsqcup \langle t_k \rangle_{k \geq 0})$ is defined if and only if all $v^\#(t_k)$, $k \geq 0$, are defined and form a chain $v^\#(t_0) \leq_s v^\#(t_1) \leq_s \ldots$, and then $v^\#(\bigsqcup \langle t_k \rangle_{k \geq 0}) = \bigsqcup_{k \geq 0} v^\#(t_k)$ (where $\bigsqcup$ on the right hand side stands for the least upper bound in the cpo $\langle |A|_s, \leq_s \rangle$).

As usual, we write $t_A(v)$ for $v^\#(t)$.

Finally, an *infinitary Σ-equation* is a triple $\langle X, t, t' \rangle$, written $\forall X \bullet t = t'$, where X is an S-sorted set of variables[8] and $t, t' \in |T^\infty_\Sigma(X)|_s$ for some $s \in S$. A continuous Σ-algebra A *satisfies* an infinitary Σ-equation $\forall X \bullet t = t'$, written $A \models_{\mathbf{CEQ}, \Sigma} \forall X \bullet t = t'$, if for all valuations $v : X \to |A|$, $t_A(v)$ and $t'_A(v)$ are both defined and equal.

We are now ready to define the institution **CEQ** of equational logic for continuous algebras:

- $\mathbf{Sign}_{\mathbf{CEQ}}$ is **AlgSig** again.
- $\mathbf{Sen}_{\mathbf{CEQ}} : \mathbf{AlgSig} \to \mathbf{Set}$ gives:
  - the set of infinitary Σ-equations for each $\Sigma \in |\mathbf{AlgSig}|$; and

[7] For simplicity, we omit the decoration of terms by their target sorts. Formally, to avoid any potential ambiguities, the definition should follow the pattern of Definition 1.4.1.

[8] For $s \in S$, the sets $X_s \subseteq \mathscr{X}$ come from a fixed vocabulary of variables as in Definition 2.1.1 and are mutually disjoint as in footnote 2.

  - the σ-translation function, mapping infinitary Σ-equations to infinitary Σ'-equations in the obvious way, for each signature morphism $\sigma: \Sigma \to \Sigma'$.

- $\mathbf{Mod_{CEQ}}: \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ gives:
  - the category $\mathbf{CAlg}(\Sigma)$ of continuous Σ-algebras and continuous Σ-homomorphisms for each $\Sigma \in |\mathbf{AlgSig}|$; and
  - the reduct functor $\_|_\sigma: \mathbf{CAlg}(\Sigma') \to \mathbf{CAlg}(\Sigma)$ defined similarly as in the case of usual (discrete) algebras for each signature morphism $\sigma: \Sigma \to \Sigma'$.
- For each $\Sigma \in |\mathbf{AlgSig}|$, the satisfaction relation $\models_{\mathbf{CEQ},\Sigma} \subseteq |\mathbf{CAlg}(\Sigma)| \times \mathbf{Sen_{CEQ}}(\Sigma)$ is the relation of satisfaction of infinitary Σ-equations by continuous Σ-algebras.

Exercise. Proceeding similarly as in the proof of the Satisfaction Lemma, show that the satisfaction condition holds for **CEQ**.

Exercise. Show that even though we have introduced only infinitary equations as sentences in **CEQ**, infinitary inequalities of the form $\forall X \bullet t \leq t'$ are expressible here as well. (HINT: $a \leq b$ iff $a \sqcup b = b$.) □

Exercise 4.1.23. For each of the institutions **INS** defined above, define formally its version $\mathbf{INS}^{der}$ based on the category of signatures with derived signature morphisms as presented in Section 1.5.2 (cf. Exercises 3.1.12 and 3.4.30). □

Example 4.1.24 (Three-valued first-order predicate logic with equality 3FOPEQ). We sketch here the institution **3FOPEQ** of three-valued first-order predicate logic with equality as an example of how the notion of an institution can cope with logical systems based on multiple truth values, where the interpretation of sentences may yield a number of values rather than just being true or false.

- $\mathbf{Sign_{3FOPEQ}}$ is the category **FOSig** of first-order signatures.
- $\mathbf{Sen_{3FOPEQ}}: \mathbf{Sign_{3FOPEQ}} \to \mathbf{Set}$ gives:
  - For each $\Theta \in |\mathbf{FOSig}|$, $\mathbf{Sen_{3FOPEQ}}(\Theta)$ is the set of sentences of the form φ **is** *tt*, φ **is** *ff*, or φ **is** *undef*, where φ is a Θ-sentence of partial first-order predicate logic with equality **PFOPEQ** (see Exercise 4.1.17).
  - For each first-order signature morphism $\theta: \Theta \to \Theta'$, we define the translation function $\mathbf{Sen_{3FOPEQ}}(\theta): \mathbf{Sen_{3FOPEQ}}(\Theta) \to \mathbf{Sen_{3FOPEQ}}(\Theta')$ in the obvious way using the translation of first-order Θ-sentences to Θ'-sentences induced by the morphism θ.
- $\mathbf{Mod_{3FOPEQ}}: \mathbf{Sign}^{op}_{\mathbf{3FOPEQ}} \to \mathbf{Cat}$ is defined as usual for first-order logic, except that operations in structures are partial functions and predicates are interpreted as *partial relations*, which for any tuple of arguments may yield one of three logical values: *tt* (for truth), *ff* (for falsity) and a "third truth value" *undef* (for undefinedness).
- Atomic formulae, propositional connectives and quantifiers may be interpreted over the three-element set of truth values $\{tt, ff, undef\}$ in a number of ways, see for example [KTB91] and references there for a discussion. Here, we adopt the following interpretation:

- Atomic definedness formulae have the expected meaning: $def(t)$ is tt if the value of t is defined, and is ff otherwise.
- Equalities are interpreted as *strict equalities*: $t = t'$ is tt if the values of t and t' are defined and equal, is ff if they are defined and different, and is *undef* otherwise.
- The propositional connectives and quantifiers are interpreted as in Kleene's calculus (cf. [KTB91]). For example, $\varphi \vee \varphi'$ is *true* if either φ or φ' is tt, is ff if both φ and φ' are ff and is *undef* otherwise.

For any $\varphi \in \mathbf{Sen}_{\mathbf{PFOPEQ}}(\Theta)$ and $M \in |\mathbf{Mod}_{\mathbf{3FOPEQ}}(\Theta)|$, this gives the *interpretation of φ in M*, $[\![\varphi]\!]_M \in \{tt, ff, undef\}$.
For each signature $\Theta \in |\mathbf{FOSig}|$, the satisfaction relation $\models_{\mathbf{3FOPEQ},\Theta} \subseteq |\mathbf{Mod}_{\mathbf{3FOPEQ}}(\Theta)| \times \mathbf{Sen}_{\mathbf{3FOPEQ}}(\Theta)$ is now defined in the obvious way: for any $M \in |\mathbf{Mod}_{\mathbf{3FOPEQ}}(\Theta)|$ and $\varphi \in \mathbf{Sen}_{\mathbf{3FOPEQ}}(\Theta)$:

- $M \models_{\mathbf{3FOPEQ},\Theta} \varphi$ **is** tt holds if and only if $[\![\varphi]\!]_M = tt$;
- $M \models_{\mathbf{3FOPEQ},\Theta} \varphi$ **is** ff holds if and only if $[\![\varphi]\!]_M = ff$; and
- $M \models_{\mathbf{3FOPEQ},\Theta} \varphi$ **is** *undef* holds if and only if $[\![\varphi]\!]_M = undef$.

Exercise. Work out all the details omitted from the above definition; notice that, in particular, model morphisms may be defined in a number of sensible ways. Then show that the satisfaction condition holds. □

Example 4.1.25 (The institution FPL of a logic for functional programs). The institution **FPL** of a logic for a simple functional programming language with a first-order monomorphic type system is defined as follows:

- A signature $\mathsf{SIG} = \langle S, \Omega, D \rangle$ consists of a set S of sort names, a family of sets of operation names $\Omega = \langle \Omega_{w,s} \rangle_{w \in S^*, s \in S}$, and a set D of *sorts with value constructors*. Elements of D have the form $\langle d, \mathscr{F} \rangle$ with $d \in S$ and $\mathscr{F} = \langle F_{w,d} \rangle_{w \in S^*}$, where $F_{w,d} \subseteq \Omega_{w,d}$ for $w \in S^*$, with no sort given more than one set of value constructors, i.e. $\langle d, \mathscr{F} \rangle, \langle d, \mathscr{F}' \rangle \in D$ implies $\mathscr{F} = \mathscr{F}'$. So SIG consists of an ordinary algebraic signature $\langle S, \Omega \rangle$ together with a set of *value constructors* for some of the sorts. Sorts with value constructors correspond to algebraic datatypes in functional programming languages. In examples we use a CASL-like notation[9], for instance:

  sort *nat* **free with** $0 \mid succ(nat)$

  adds *nat* to S, $0 : nat$ and $succ : nat \to nat$ to Ω, and $\langle nat, \{0 : nat, succ : nat \to nat\} \rangle$ to D. We assume for convenience that each **FPL** signature SIG contains the sort *bool* with value constructors *true* and *false*:

  sort *bool* **free with** *true* | *false*

[9] CASL notation: this would be written **free type** $nat ::= 0 \mid succ(nat)$ in CASL.

- A model over a signature $\mathsf{SIG} = \langle S, \Omega, D\rangle$ is a partial $\langle S, \Omega\rangle$-algebra A such that for each set[10] of sorts with value constructors $\{\langle d_1, \mathscr{F}_1\rangle, \ldots, \langle d_n, \mathscr{F}_n\rangle\} \subseteq D$, for $1 \leq i \leq n$, each value constructor in $\mathscr{F}_i$ is total and each element $a \in |A|_{d_i}$ is uniquely constructed from the values in $|A|$ of sorts other than $d_1, \ldots, d_n$ using the value constructors in $\mathscr{F}_1 \cup \cdots \cup \mathscr{F}_n$; that is, $\langle |A|_{d_i}\rangle_{1 \leq i \leq n}$ is freely generated by $\mathscr{F}_1 \cup \cdots \cup \mathscr{F}_n$ from the carriers of the other sorts in A.
  We assume that all **FPL**-models interpret the sort *bool* and its constructors *true* and *false* in some standard way.
  A SIG-morphism between SIG-models A and B is an $\langle S, \Omega\rangle$-homomorphism between A and B viewed as partial $\langle S, \Omega\rangle$-algebras. It is *strong* if it is strong when viewed as a homomorphism between partial algebras, see Definition 2.7.31.
- The set $|T_{\mathsf{SIG}}(X)|$ of **FPL**-terms over $\mathsf{SIG} = \langle S, \Omega, D\rangle$ with variables X and their interpretation in an **FPL**-model A are defined by extending the usual definition of terms over $\langle S, \Omega\rangle$ and their interpretation by the following additional functional programming constructs (local recursive function definitions and pattern-matching case analysis, respectively):
  - **let fun** $f(x_1{:}s_1, \ldots, x_n{:}s_n){:}s' = t'$ **in** t is an **FPL**-term of sort s with variables in X if:
    - $s_1, \ldots, s_n, s' \in S$;
    - t' is an **FPL**-term of sort s' over SIG extended by $f{:}s_1 \times \cdots \times s_n \to s'$ with variables in $X \cup \{x_1{:}s_1, \ldots, x_n{:}s_n\}$; and
    - t is an **FPL**-term of sort s over SIG extended by $f{:}s_1 \times \cdots \times s_n \to s'$ with variables in X.

    The value of such a term under a valuation $v{:}X \to |A|$ is determined as follows:
    - extend A to give an algebra $\widehat{A}$ by interpreting $f{:}s_1 \times \cdots \times s_n \to s'$ as the least-defined partial function $f_{\widehat{A}}$ such that for all $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$, the value of $f_{\widehat{A}}(a_1, \ldots, a_n)$ is the same as the value of t' in $\widehat{A}$ under v modified by mapping x_1 to a_1 and $\ldots$ and x_n to a_n, whenever the latter is defined.[11]
    - the resulting value is then the value of t in $\widehat{A}$ under v.
  - **case** t **of** pat_1=>t_1 | $\cdots$ | pat_n=>t_n is an **FPL**-term of sort s with variables in X if:
    - t is an **FPL**-term of some sort s' over SIG with variables in X;
    - for each $1 \leq j \leq n$, pat_j is a *pattern* over SIG of sort s', where a pattern is an $\langle S, \Omega\rangle$-term containing only variables and value constructors, with no repeated variable occurrences; and

[10] This definition is complicated because of the possible presence of mutually dependent sorts with value constructors. **Exercise:** Check that imposing the same requirement for each sort with value constructors separately is more permissive and would not capture the intended meaning. Check also that it would be sufficient to consider only maximal sets of sorts with values constructors that are mutually dependent.

[11] The fact that this unambiguously defines $f_{\widehat{A}}$, and that $f_{\widehat{A}}$ can be equivalently given via the natural operational semantics of recursively-defined functions, is a standard result of denotational semantics, see for instance [Sch86].

  - for each $1 \leq j \leq n$, t_j is an **FPL**-term of sort s with variables in the set X extended by the variables of pat_j.

  The value of such a term under a valuation $v: X \to |A|$ is determined as follows:
  - obtain the value a of t in A under v;
  - find the least j such that a *matches* pat_j yielding a valuation v' of the variables in pat_j, where matching a value against a pattern proceeds as follows:
    - a variable x is matched by any value a, yielding a valuation $\{x \mapsto a\}$;
    - a pattern $f(p_1, \ldots, p_m)$ is matched by a yielding v' iff[12] $a = f_A(a_1, \ldots, a_m)$ and each p_i $(1 \leq i \leq m)$ is matched by a_i yielding v'_i, with $v' = v'_1 \cup \cdots \cup v'_m$;
  - the resulting value is that of t_j in A under the extension of v by v' if such a j exists; otherwise, the resulting value is undefined.

- Sentences over SIG are first-order sentences built over atomic formulae which are equalities between **FPL**-terms over SIG of the same sort and definedness assertions for such terms. Interpretation of **FPL**-terms in a model determines satisfaction of such sentences as in **PFOEQ**, see Exercises 4.1.17 and 4.1.18. (Recall that **PFOEQ** uses *strong* equality, see Section 2.7.4.)
  For convenience, we introduce *function definitions* of the form

  fun $f(x_1{:}s_1, \ldots, x_n{:}s_n){:}s = t$

  to abbreviate the formula

  $\forall x_1{:}s_1, \ldots, x_n{:}s_n$
  - $f(x_1, \ldots, x_n) =$ **let fun** $f(x_1{:}s_1, \ldots, x_n{:}s_n){:}s = t$ **in** $f(x_1, \ldots, x_n)$.

  To make the scopes of identifiers clearer, this can be rewritten using a new operation name g as

  $\forall x_1{:}s_1, \ldots, x_n{:}s_n$
  - $f(x_1, \ldots, x_n) =$ **let fun** $g(x_1{:}s_1, \ldots, x_n{:}s_n){:}s = t'$ **in** $g(x_1, \ldots, x_n)$

  where t' is the result of replacing f by g in t. Such a recursive function definition is different from the equality $f(x_1, \ldots, x_n) = t$: for instance, $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ always holds while **fun** $f(x_1{:}s_1, \ldots, x_n{:}s_n){:}s = f(x_1, \ldots, x_n)$ holds only when f is totally undefined.
- Given $\mathsf{SIG} = \langle S, \Omega, D \rangle$ and $\mathsf{SIG}' = \langle S', \Omega', D' \rangle$, an **FPL** signature morphism $\delta: \mathsf{SIG} \to \mathsf{SIG}'$ is a derived signature morphism $\delta: \langle S, \Omega \rangle \to \langle S', \Omega' \rangle$ (using **FPL**-terms in place of ordinary terms in Definition 1.5.13), such that for each $\langle d, \mathscr{F} \rangle \in D$, we have $\langle \delta(d), \mathscr{F}' \rangle \in D'$ such that δ restricted to $\mathscr{F}$ is determined by a bijection from $\mathscr{F}$ to $\mathscr{F}'$.
  We require all **FPL** signature morphisms to preserve the sort *bool* and its constructors *true* and *false*.
  Such signature morphisms go well beyond the usual renaming of sort and operation names; here we allow (non-constructor) operations to be mapped to

[12] This uniquely determines a result because non-variable patterns are of sorts that are freely generated by the value constructors and there are no repeated occurrences of variables in patterns.

complicated terms involving programming constructs like recursion and pattern-matching case analysis. This will be used in Chapters 6–9 to give examples, starting with Example 6.1.6, that suggest how programs fit into the overall specification and development framework.

Such a signature morphism determines a translation of SIG-sentences to SIG′-sentences in the usual manner,[13] and the same for the reduct from SIG′-models to SIG-models. Moreover, the satisfaction condition holds.

Exercise. Complete the above definition and prove the satisfaction condition. □

Exercise 4.1.26. The functional programming constructs used above are inspired by those in Standard ML [Pau96]. Add more constructs from Standard ML to the definition of **FPL**. Try adding type definitions, polymorphism, higher-order functions, exceptions.

It is easy to add built-in types other than *bool* by basing the definition of **FPL** on an arbitrary algebra DT as in **IMP** (Example 4.1.32 below). □

Exercise 4.1.27. Mutual recursion need not be added explicitly since it is already expressible using local definitions of recursive functions. Show how. HINT: It may be necessary to resort to copying function definitions, to make each function available for the definitions of the others. □

Exercise 4.1.28. Consider an **FPL**-signature SIG containing a sort s that is freely generated by value constructors from other such sorts. Show how an equality operation $eq_s : s \times s \to bool$ may be defined using a recursive function definition with pattern-matching case analysis. Use this to view conditionals of the form

if $t_1 = t_2$ **then** t **else** t'

(where t_1, t_2 are SIG-terms of sort s, and t, t' have the same sort) as an abbreviation for

let fun $eq_s(x{:}s, y{:}s){:}bool = \ldots$ **in case** $eq_s(t_1, t_2)$ **of** *true*=>t | *false*=>t' □

Exercise 4.1.29. One could also introduce a conditional of the form **if** φ **then** t **else** t' where φ is a formula. Spell out the details. This would be unusual as a programming construct because branching is controlled by an arbitrary logical formula, allowing terms that would be problematic from a programming point of view, such as **if** $def(t)$ **then** t' **else** t'' and **if** $\forall x{:}s \bullet t_1 = t_2$ **then** t' **else** t''. Note that the meaning of such a conditional would be different from the one introduced in Exercise 4.1.28 when the check for equality involves a term with no defined value. □

---

[13] Care is required to avoid unintended clashes of **let**-bound operation names in SIG-terms with operation names in SIG′. To avoid consequent problems with functoriality of sentence translation, we can regard **FPL**-terms as being defined up to renaming of **let**-bound operation names.

Moreover, as in **FOPEQ** (see Example 4.1.12), care is needed with the treatment of bound variables (which now also include variables in patterns and formal parameters in **let**-bound operation definitions), cf. footnote 6.

Exercise 4.1.30. While **FPL** involves constructs borrowed from functional programming languages, it puts them in a logical context involving equality, logical connectives and quantifiers, which results in sentences capable not only of defining functions, but also of specifying their properties. Identify the "programming part" of **FPL** by defining its "subinstitution" **FProg** with the same signatures and models, but with sets of sentences restricted to function definitions (with satisfaction relations inherited from **FPL** as well). As function definitions may not be closed under translation along arbitrary (derived) signature morphisms in **FPL**, restrict the class of signature morphisms in **FProg** to the standard morphisms, where operation names are mapped to operation names rather than to arbitrary terms. □

Exercise 4.1.31. **FPL**, and its programming part **FProg**, relate to eager functional programming languages like Standard ML because partial functions are required to be strict. Formulate an analogous institution for lazy functional programming as in Haskell. □

The institutions **FPL** and **FProg** will be used in the sequel to present examples that are meant to appeal to the reader's programming intuition. Later on, the connection with functional programming will be further enhanced by introducing notations for defining ML-style modules in **FPL** (see Example 6.1.9 and Exercise 7.3.5 below).

Example 4.1.32 (The institution IMP of a simple imperative language). The institution **IMP** of an imperative programming language with simple type definitions is parameterised by an algebra DT on a signature Σ_{DT} of primitive (built-in) data types and functions of the language. The components of $\mathbf{IMP}_{DT}$ are defined as follows:

- A signature $\Pi = \langle T, P \rangle$ consists of a set T of type names and a set P of functional procedure names with types of the form $s_1, \ldots, s_n \to s$, where each of $s_1, \ldots, s_n, s$ is either a sort in Σ_{DT} or a type name in T. The names in T and P are distinct from those in Σ_{DT}. Thus $\Pi \cup \Sigma_{DT}$ is an algebraic signature — we will denote it by Π_{DT}. Signature morphisms map type names to type names and procedure names to procedure names preserving their types.
- There are two kinds of sentences over a signature $\Pi = \langle T, P \rangle$.
  First, sentences can be type definitions of the form

  type $s =$ *type-expr*

  where $s \in T$ is a type name and *type-expr* is a type expression in a simple language of types built over the sorts in Σ_{DT} and a unit type `unit` using the operators $+$ (disjoint union) and $\times$ (Cartesian product). The type expression *type-expr* may contain the type name s as well, which provides for recursive type definitions.[14]
  Second, sentences can be procedure definitions of the form

[14] Other type names from T are excluded, to prevent mutual recursion in type definitions — with some extra work this restriction can be removed.

proc $p(x_1{:}s_1,\ldots,x_n{:}s_n) = \textit{while-program}$; **result** $\textit{expr}{:}s$

where $p{:}s_1,\ldots,s_n \to s$ is a procedure name in P, *expr* is a Π_{DT}-term (with variables) of sort s, and *while-program* is a statement in a deterministic programming language over the built-in data types and functions given in DT (*while-program* may be empty, and so the program part of a procedure body may be omitted). We assume that the usual iterative program constructions are provided: sequential statements, conditionals and while loops. This requires that Σ_{DT} contains the sort *bool* with $|DT|_{bool} = \{tt, ff\}$. The basic statements are well-typed assignments (of expression values to formal parameters or variables scoped within each procedure body).
Expressions may use projections $\texttt{proj}_1(v)$ and $\texttt{proj}_2(v)$ for values v of product types of the form $s_1 \times s_2$, and pairing $\langle v_1, v_2\rangle$ to build values of product types, as well as boolean tests $\texttt{is-in}_1(v)$ and $\texttt{is-in}_2(v)$ for values v of union types of the form $s_1 + s_2$ and the constant $\langle\rangle$ of type `unit` denoting the only element of this type. The usual coercions between union types and their component types may also be used. With a bit of additional complication we can also allow expressions to contain (recursive) procedure calls.

- A model M over a signature $\Pi = \langle T, P\rangle$ has a carrier set $|M|_s$ for each $s \in T$. We write $|M|_s$ for $|DT|_s$ if s is a sort name in Σ_{DT}.
  We have the usual notion of *state*, where each state maps formal parameters and variables to values of their sorts in M, or marks them as undefined. An obvious operational semantics may be given that determines, for each statement and state, a sequence of states that formally captures the execution of that statement starting in that state.
  Then, M assigns to each procedure name $p{:}s_1,\ldots,s_n \to s$ in P and every sequence $v_1 \in |M|_{s_1},\ldots,v_n \in |M|_{s_n}$ of (actual parameter) values a formal execution which has one of the following forms:

  (*Successful termination*): a finite sequence of states and a value $v \in |M|_s$;
  (*Unsuccessful termination*): a finite sequence of states; or
  (*Divergence*): an infinite sequence of states.

  Given any such model M, for any procedure name $p{:}s_1,\ldots,s_n \to s$ in P we get a partial function $p_M{:}|M|_{s_1} \times \cdots \times |M|_{s_n} \to |M|_s$.
  The models defined in this way form a discrete category.
- For any signature $\Pi = \langle T, P\rangle$ and Π-model M:
  - M satisfies a Π-sentence of the form

    type $s = \textit{type-expr}$

    if $|M|_s$ is the least set $\mathbb{D}$ such that $\mathbb{D}$ is the value of the type expression *type-expr* in which the type name s is interpreted as $\mathbb{D}$ and sort names s' in Σ_{DT} are interpreted as $|DT|_{s'}$.
  - M satisfies a Π-sentence of the form

    proc $p(x_1{:}s_1,\ldots,x_n{:}s_n) = \textit{while-program}$; **result** $\textit{expr}{:}s$

if for all $v_1 \in |M|_{s_1}, \ldots, v_n \in |M|_{s_n}$, $M(p)(v_1, \ldots, v_n)$ is the formal execution of the statement *while-program* starting in the state $\{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$, and if the execution terminates successfully in a state in which *expr* has a defined value then $M(p)(v_1, \ldots, v_n)$ contains this value as well.

Exercise. Complete the above definition and prove the satisfaction condition. □

Exercise 4.1.33. Sentences in **IMP** are essentially programs; they provide no means of writing loose specifications. Add sentences of **PFOPEQ** for specifying properties of the procedures of **IMP** viewed as partial functions. A different way of achieving a similar effect will be presented in Examples 10.1.9, 10.1.14 and 10.1.17. □

Example 4.1.34 (The institution CDIAG of commutative diagrams). The following example is of a rather non-standard character. We present a simple logical system for stating that certain diagrams in a category with named objects and morphisms commute. Sentences of the logical system allow one to require that morphisms produced by composition of series of (named) morphisms coincide.

- The category of signatures in **CDIAG** is the category **Graph** of graphs (see Definition 3.2.36).
- A *path equation* in a graph G is a pair of paths in G with the same sources and targets, respectively. For any graph G (a signature in $\mathbf{Sign}_{\mathbf{CDIAG}}$), G-sentences in **CDIAG** are sets of path equations in G.
- A model over a graph G is a (small) category $\mathbf{C}$ with a diagram D of "shape" G, i.e. (via Exercise 3.4.21) a functor $D\colon \mathbf{Path}(G) \to \mathbf{C}$. For any two G-models $D1\colon \mathbf{Path}(G) \to \mathbf{C1}$ and $D2\colon \mathbf{Path}(G) \to \mathbf{C2}$, a *$G$-morphism* in $\mathbf{Mod}_{\mathbf{CDIAG}}(G)$ from $D1$ to $D2$ is a functor $\mathbf{F}\colon \mathbf{C1} \to \mathbf{C2}$ such that $D1;\mathbf{F} = D2$.
- For any G-model $D\colon \mathbf{Path}(G) \to \mathbf{C}$, a path p from s to t in G determines a morphism $D(p)\colon D(s) \to D(t)$ in $\mathbf{C}$. We say that a G-model $D\colon \mathbf{Path}(G) \to \mathbf{C}$ *satisfies* a path equation $\langle p, q \rangle$ if $D(p) = D(q)$. A G-model satisfies a G-sentence Φ if it satisfies all path equations $\varphi \in \Phi$.

Exercise. Complete the definition and prove the satisfaction condition for **CDIAG**.

Exercise. Reformulate the above definitions so that a sentence over a graph G would be a subdiagram of G used to denote the set of path equations in G which make the subdiagram commute. □

The last few examples show that the notion of institution covers much more than what one usually connects with the concept of a logical system.

The next two examples are perhaps even more unusual: we show that the definition of an institution does not restrict the sentences of a logic to be syntactic objects, and does not force models to provide semantic domains and operations used to determine the meanings of the syntactic objects. Thus, the notion of an institution covers systems in which such a distinction is entirely blurred.

Example 4.1.35. Consider an arbitrary category **Sign** and functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$. We think of **Sign** as a category of signatures and of **Mod** as yielding categories

of models and reduct functors. To be cautious about foundations, we should make sure that **Mod** yields only small categories.

We can now define an institution $\mathbf{INS}^{\mathrm{Sen}(\mathbf{Mod})}$ where "sentences" are classes of models:

- The category of signatures of $\mathbf{INS}^{\mathrm{Sen}(\mathbf{Mod})}$ is **Sign**.
- The "sentence" functor of $\mathbf{INS}^{\mathrm{Sen}(\mathbf{Mod})}$ is defined as follows:
  - For any signature $\Sigma \in |\mathbf{Sign}|$, a Σ-"sentence" of $\mathbf{INS}^{\mathrm{Sen}(\mathbf{Mod})}$ is a collection $\mathscr{M} \subseteq |\mathbf{Mod}(\Sigma)|$ of Σ-models.
  - For any signature morphism $\sigma: \Sigma \to \Sigma'$, the σ-translation of any Σ-"sentence" $\mathscr{M} \subseteq |\mathbf{Mod}(\Sigma)|$ to a Σ'-"sentence" $\sigma(\mathscr{M}) \subseteq |\mathbf{Mod}(\Sigma')|$ is defined as the coimage of $\mathscr{M}$ w.r.t. the σ-reduct functor, i.e. $\sigma(\mathscr{M}) = \{M' \in |\mathbf{Mod}(\Sigma')| \mid \mathbf{Mod}(\sigma)(M') \in \mathscr{M}\}$.
- The model functor of $\mathbf{INS}^{\mathrm{Sen}(\mathbf{Mod})}$ is **Mod**.
- For each signature Σ, the Σ-satisfaction relation of $\mathbf{INS}^{\mathrm{Sen}(\mathbf{Mod})}$ is just the membership relation: for any Σ-model $M \in |\mathbf{Mod}(\Sigma)|$ and Σ-"sentence" $\mathscr{M} \subseteq |\mathbf{Mod}(\Sigma)|$, $M \models_{\mathbf{INS}^{\mathrm{Sen}(\mathbf{Mod})},\Sigma} \mathscr{M}$ if and only if $M \in \mathscr{M}$.

Exercise. Complete the definition and check the satisfaction condition. □

Example 4.1.36. Consider an arbitrary category **Sign** and functor $\mathbf{Sen}: \mathbf{Sign} \to \mathbf{Set}$. We think of **Sign** as a category of signatures and of **Sen** as yielding sets of sentences and their translations.

We can now define an institution $\mathbf{INS}^{\mathrm{Mod}(\mathbf{Sen})}$ where "models" are sets of sentences:

- The category of signatures of $\mathbf{INS}^{\mathrm{Mod}(\mathbf{Sen})}$ is **Sign**.
- The sentence functor of $\mathbf{INS}^{\mathrm{Mod}(\mathbf{Sen})}$ is **Sen**.
- The "model" functor of $\mathbf{INS}^{\mathrm{Mod}(\mathbf{Sen})}$ is defined as follows:
  - For any signature $\Sigma \in |\mathbf{Sign}|$, a Σ-"model" of $\mathbf{INS}^{\mathrm{Mod}(\mathbf{Sen})}$ is a set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences. The category of Σ-"models" is just the preorder category where the set of all such subsets is ordered by inclusion.
  - For any signature morphism $\sigma: \Sigma \to \Sigma'$, the σ-reduct functor of $\mathbf{INS}^{\mathrm{Mod}(\mathbf{Sen})}$ from the category of Σ'-"models" to the category of Σ-"models" maps any Σ'-"model" $\Phi' \subseteq \mathbf{Sen}(\Sigma')$ to its coimage $\{\varphi \in \mathbf{Sen}(\Sigma) \mid \mathbf{Sen}(\sigma)(\varphi) \in \Phi'\} \subseteq \mathbf{Sen}(\Sigma)$; this obviously extends to a functor between the preorder categories of Σ'- and Σ-"models".
- For each signature Σ, the Σ-satisfaction relation of $\mathbf{INS}^{\mathrm{Mod}(\mathbf{Sen})}$ is (the inverse of) the membership relation: for any Σ-"model" $\Phi \subseteq \mathbf{Sen}(\Sigma)$ and Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$, $\Phi \models_{\mathbf{INS}^{\mathrm{Mod}(\mathbf{Sen})},\Sigma} \varphi$ if and only if $\varphi \in \Phi$.

Exercise. Complete the definition and check the satisfaction condition. □

Let us complete this list of examples by pointing out that the definition of institution admits a number of trivial situations:

Example 4.1.37 (Trivial institutions).

- Recall that **0** is the empty category. Hence, there is a unique (empty) functor from **0** to **Set** and a unique (empty) functor from $\mathbf{0}^{op} = \mathbf{0}$ to **Cat**. Together with the empty family of relations, they form an empty institution (no signatures, hence no sentences and no models).
- Given any category **Sign** and functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$, a trivial institution with signatures **Sign**, with models given by **Mod**, and with no sentences may be constructed. Formally, the sentences of this institution are given by the functor $\mathbf{Sen}_{\varnothing}\colon \mathbf{Sign} \to \mathbf{Set}$ which yields the empty set for each signature.
- Given any category **Sign** and functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$, a trivial institution with signatures **Sign**, with sentences given by **Sen**, and with no models may be constructed. Formally, the models of this institution are given by the functor $\mathbf{Mod}_{\mathbf{0}}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$ which yields the empty category for each signature.
- Given any category **Sign** and functors $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$ and $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$, two trivial institutions with signatures **Sign**, with sentences given by **Sen**, and with models given by **Mod** may be constructed. One is obtained by making all sentences false in all models, that is by defining each satisfaction relation to be empty. The other is obtained by making all sentences hold in all models, that is by defininining each satisfaction relation to be total (i.e., for each $\Sigma \in |\mathbf{Sign}|$, $\models_{\Sigma} = |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$). □

4.1.2 Constructing institutions

In the examples of the previous subsection, each of the institutions was constructed "from scratch" by explicitly defining its signatures, sentences, models and satisfaction relations. This is often a rather tedious task (we have simplified it in many cases by referring to the standard definitions) and then checking the satisfaction condition is not always easy. In this subsection we will give some examples of constructions leading from an institution to a more complex one. The complexity added by the construction does not necessarily imply that the institution so obtained has any extra "expressive power". We start with some examples of "formal juggling" with institution components, very much in the spirit of Examples 4.1.35 and 4.1.36, and only then show how adding propositional connectives to a logic may be viewed as a construction of a new institution from an existing one.

Example 4.1.38. Sets of sentences of any institution may be regarded as single sentences (with the obvious "conjunctive" interpretation).

For any institution **INS** define the institution $\mathbf{INS}^{\wedge}$ of sets of **INS**-sentences as follows:

- The category of $\mathbf{INS}^{\wedge}$-signatures is the same as the category **Sign** of **INS**-signatures.
- The sentence functor $\mathbf{Sen}_{\mathbf{INS}^{\wedge}}$ is defined as follows:

  - For any signature $\Sigma \in |\mathbf{Sign}|$, $\mathbf{Sen}_{\mathbf{INS}^\wedge}(\Sigma)$ is the set of all sets $\Phi \subseteq \mathbf{Sen}_{\mathbf{INS}}(\Sigma)$ of Σ-sentences in **INS**.
  - For any signature morphism $\sigma: \Sigma \to \Sigma'$, the translation of a Σ-sentence Φ in $\mathbf{INS}^\wedge$ is its image w.r.t. the σ-translation function in **INS**: $\mathbf{Sen}_{\mathbf{INS}^\wedge}(\sigma)(\Phi) = \{\mathbf{Sen}_{\mathbf{INS}}(\sigma)(\varphi) \mid \varphi \in \Phi\} \subseteq \mathbf{Sen}_{\mathbf{INS}}(\Sigma')$.

- The model functor of $\mathbf{INS}^\wedge$ is the same as the model functor $\mathbf{Mod}: \mathbf{Sign}^{op} \to \mathbf{Cat}$ of **INS**.
- For any signature $\Sigma \in |\mathbf{Sign}|$, the satisfaction relation of $\mathbf{INS}^\wedge$ gives the conjunctive interpretation of (sets of) sentences: for any Σ-model $M \in |\mathbf{Mod}(\Sigma)|$ and Σ-sentence $\Phi \subseteq \mathbf{Sen}_{\mathbf{INS}}(\Sigma)$, $M \models_{\mathbf{INS}^\wedge,\Sigma} \Phi$ if and only if for all $\varphi \in \Phi$, $M \models_{\mathbf{INS},\Sigma} \varphi$. □

Example 4.1.39. Signatures of any institution may be enriched to incorporate sentences which restrict the class of models considered over the given signature.

For any institution **INS** define the institution $\mathbf{INS}^{\mathbf{Sign}^+}$ with signatures enriched by sentences as follows:

- Signatures of $\mathbf{INS}^{\mathbf{Sign}^+}$ are pairs $\langle \Sigma, \Phi \rangle$, where $\Sigma \in |\mathbf{Sign}_{\mathbf{INS}}|$ is an **INS**-signature and $\Phi \subseteq \mathbf{Sen}_{\mathbf{INS}}(\Sigma)$ is a set of Σ-sentences. Then, an $\mathbf{INS}^{\mathbf{Sign}^+}$-signature morphism $\sigma: \langle \Sigma, \Phi \rangle \to \langle \Sigma', \Phi' \rangle$ is a signature morphism $\sigma: \Sigma \to \Sigma'$ in $\mathbf{Sign}_{\mathbf{INS}}$ such that for all $\varphi \in \Phi$, $\sigma(\varphi) \in \Phi'$. This defines a category $\mathbf{Sign}_{\mathbf{INS}^{\mathbf{Sign}^+}}$ of $\mathbf{INS}^{\mathbf{Sign}^+}$-signatures (with composition inherited from $\mathbf{Sign}_{\mathbf{INS}}$).
- Sentences of $\mathbf{INS}^{\mathbf{Sign}^+}$ are the same as **INS**-sentences: for any $\mathbf{INS}^{\mathbf{Sign}^+}$-signature $\langle \Sigma, \Phi \rangle$, $\mathbf{Sen}_{\mathbf{INS}^{\mathbf{Sign}^+}}(\langle \Sigma, \Phi \rangle) = \mathbf{Sen}_{\mathbf{INS}}(\Sigma)$, with the translation functions inherited from **INS** as well.
- Models of $\mathbf{INS}^{\mathbf{Sign}^+}$ are again the same as models of **INS**; we consider, however, only those models that satisfy the sentences in the given signature. For any $\mathbf{INS}^{\mathbf{Sign}^+}$-signature $\langle \Sigma, \Phi \rangle$, $\mathbf{Mod}_{\mathbf{INS}^{\mathbf{Sign}^+}}(\langle \Sigma, \Phi \rangle)$ is the full subcategory of $\mathbf{Mod}_{\mathbf{INS}}(\Sigma)$ consisting of all Σ-models (in **INS**) that satisfy (according to $\models_{\mathbf{INS},\Sigma}$) all the sentences in Φ. The reduct functors are again inherited from **INS**.
- The satisfaction relations of $\mathbf{INS}^{\mathbf{Sign}^+}$ are inherited from **INS**.

Exercise. Spell out all the details of the above definition. In particular, check that the reduct functors of the new institution $\mathbf{INS}^{\mathbf{Sign}^+}$ are well-defined (cf. Fact 4.2.24 below). □

Example 4.1.40. For any institution, we can enlarge its categories of models by considering models over extended signatures.

For any institution **INS**, define the institution $\mathbf{INS}^{\mathbf{Mod}^+}$ with categories of models containing models over extended signatures as follows:

- The category of $\mathbf{INS}^{\mathbf{Mod}^+}$-signatures is the category **Sign** of **INS**-signatures.
- The sentence functor of $\mathbf{INS}^{\mathbf{Mod}^+}$ is the sentence functor $\mathbf{Sen}: \mathbf{Sign} \to \mathbf{Set}$ of **INS**.
- The model functor $\mathbf{Mod}_{\mathbf{INS}^{\mathbf{Mod}^+}}: \mathbf{Sign}^{op} \to \mathbf{Cat}$ is defined as follows:

  - For any signature $\Sigma \in |\mathbf{Sign}|$, a Σ-model of $\mathbf{INS}^{\mathbf{Mod}^+}$ is an **INS**-model over an extension of the signature Σ. Formally: a Σ-model in $\mathbf{INS}^{\mathbf{Mod}^+}$ is a pair $\langle \sigma: \Sigma \to \Sigma', M' \in |\mathbf{Mod}_{\mathbf{INS}}(\Sigma')| \rangle$. A Σ-model morphism between two such Σ-models is again a pair $\langle \sigma', f \rangle: \langle \sigma_1: \Sigma \to \Sigma'_1, M'_1 \in |\mathbf{Mod}_{\mathbf{INS}}(\Sigma'_1)| \rangle \to \langle \sigma_2: \Sigma \to \Sigma'_2, M'_2 \in |\mathbf{Mod}_{\mathbf{INS}}(\Sigma'_2)| \rangle$, which consists of an **INS**-signature morphism $\sigma': \Sigma'_1 \to \Sigma'_2$ such that $\sigma_1; \sigma' = \sigma_2$ and a model morphism $f: M'_1 \to \mathbf{Mod}_{\mathbf{INS}}(\sigma')(M'_2)$ in $\mathbf{Mod}_{\mathbf{INS}}(\Sigma'_1)$.
  - For any signature morphism $\sigma: \Sigma_1 \to \Sigma_2$, the σ-reduct functor $\mathbf{Mod}_{\mathbf{INS}^{\mathbf{Mod}^+}}(\sigma)$ maps any Σ_2-model $\langle \sigma_2: \Sigma_2 \to \Sigma'_2, M'_2 \in |\mathbf{Mod}_{\mathbf{INS}}(\Sigma'_2)| \rangle$ to the Σ_1-model $\langle \sigma; \sigma_2: \Sigma_1 \to \Sigma'_2, M'_2 \in |\mathbf{Mod}_{\mathbf{INS}}(\Sigma'_2)| \rangle$. On model morphisms, $\mathbf{Mod}_{\mathbf{INS}^{\mathbf{Mod}^+}}(\sigma)$ is the identity.

- For each signature $\Sigma \in |\mathbf{Sign}|$, the Σ-satisfaction relation of $\mathbf{INS}^{\mathbf{Mod}^+}$ is determined by the Σ-satisfaction relation of **INS**: for any Σ-model $\langle \sigma: \Sigma \to \Sigma', M' \in |\mathbf{Mod}_{\mathbf{INS}}(\Sigma')| \rangle$ and Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$, $\langle \sigma, M' \rangle \models_{\mathbf{INS}^{\mathbf{Mod}^+}, \Sigma} \varphi$ if and only if $M' \models_{\mathbf{INS}, \Sigma'} \mathbf{Sen}(\sigma)(\varphi)$, which by the satisfaction condition for **INS** is equivalent to $\mathbf{Mod}_{\mathbf{INS}}(\sigma)(M') \models_{\mathbf{INS}, \Sigma} \varphi$.

Exercise. Complete the definition and check the satisfaction condition. Try to express the construction of the categories of models of $\mathbf{INS}^{\mathbf{Mod}^+}$ using the flattening construction for indexed categories (Definition 3.4.58) and the machinery of comma categories (Definition 3.4.49). □

Example 4.1.41. For any institution **INS** define the institution $\mathbf{INS}^{prop}$ by closing the sets of its sentences under propositional connectives (with the usual interpretation) as follows:

- The category of signatures of $\mathbf{INS}^{prop}$ is just the category **Sign** of **INS**-signatures.
- The sentence functor $\mathbf{Sen}_{\mathbf{INS}^{prop}}: \mathbf{Sign} \to \mathbf{Set}$ is defined as follows:
  - For any signature $\Sigma \in |\mathbf{Sign}|$, $\mathbf{Sen}_{\mathbf{INS}^{prop}}(\Sigma)$ is the least set that contains all of the Σ-sentences of **INS** and two special sentences true and false, and is closed under the usual propositional connectives as introduced in Example 4.1.9, that is, if $\varphi, \varphi' \in \mathbf{Sen}_{\mathbf{INS}^{prop}}(\Sigma)$ then also $\varphi \vee \varphi' \in \mathbf{Sen}_{\mathbf{INS}^{prop}}(\Sigma)$, $\neg\varphi \in \mathbf{Sen}_{\mathbf{INS}^{prop}}(\Sigma)$, $\varphi \wedge \varphi' \in \mathbf{Sen}_{\mathbf{INS}^{prop}}(\Sigma)$, and $\varphi \Rightarrow \varphi' \in \mathbf{Sen}_{\mathbf{INS}^{prop}}(\Sigma)$.[15]
  - For any signature morphism $\sigma: \Sigma \to \Sigma'$, the σ-translation function $\mathbf{Sen}_{\mathbf{INS}^{prop}}(\sigma)$ coincides with $\mathbf{Sen}_{\mathbf{INS}}(\sigma)$ on $\mathbf{Sen}_{\mathbf{INS}}(\Sigma)$ and preserves the propositional connectives in the new sentences in the obvious way.

- The model functor of $\mathbf{INS}^{prop}$ is the model functor $\mathbf{Mod}: \mathbf{Sign}^{op} \to \mathbf{Cat}$ of **INS**.
- For each signature $\Sigma \in |\mathbf{Sign}|$, the Σ-satisfaction relation of $\mathbf{INS}^{prop}$ is just the same as the Σ-satisfaction relation of **INS** for sentences in $\mathbf{Sen}_{\mathbf{INS}}(\Sigma)$ and then, for any Σ-model $M \in |\mathbf{Mod}(\Sigma)|$, for the sentences built using the propositional connectives, the satisfaction of such sentences in M is defined inductively as in Example 4.1.9.

[15] The remarks in footnote 4 apply as appropriate.

Exercise. Show how **PROP**, the institution of propositional logic (see Example 4.1.9) arises as the propositional closure of a simple institution with propositional variables as the only sentences. □

In Section 4.4.2 below we define yet another similar construction on institutions by showing how quantifiers may be introduced.

The constructions described in the examples above may naturally be viewed as extensions of the original institution — this will be made formal in Section 10.2, cf. Example 10.2.5. In Section 10.3 we will discuss how such extensions may be combined using the limit construction in a suitable category of institutions.

These examples are about adding new sentences built using logical connectives to an institution. The new sentences are added, even if the connectives were already expressible in the following sense:

Definition 4.1.42. The institution **INS** *has negation* if for every signature $\Sigma \in |\mathbf{Sign}|$ and Σ-sentence φ, there exists a Σ-sentence ψ such that for every Σ-model M, $M \models_\Sigma \varphi$ iff $M \not\models_\Sigma \psi$. Any such ψ may be referred to as $\neg\varphi$.

The properties of *having conjunction*, *having disjunction* and *having implication* are defined in the analogous way, and similarly for *having truth*, *having falsity*, *having infinitary conjunction* etc. □

Exercise 4.1.43. Suppose that the institution **INS** has negation. Using the satisfaction condition, show that for every signature morphism $\sigma: \Sigma \to \Sigma'$ and Σ-sentence φ, $\neg\sigma(\varphi)$ may be taken to be $\sigma(\neg\varphi)$. Show a similar property for the other connectives. □

Example 4.1.44. For any institutions $\mathbf{INS}_1 = \langle \mathbf{Sign}_1, \mathbf{Sen}_1, \mathbf{Mod}_1, \langle \models_{1,\Sigma_1} \rangle_{\Sigma_1 \in |\mathbf{Sign}_1|} \rangle$ and $\mathbf{INS}_2 = \langle \mathbf{Sign}_2, \mathbf{Sen}_2, \mathbf{Mod}_2, \langle \models_{2,\Sigma_2} \rangle_{\Sigma_2 \in |\mathbf{Sign}_2|} \rangle$, their *sum* $\mathbf{INS}_1 + \mathbf{INS}_2$ puts $\mathbf{INS}_1$ and $\mathbf{INS}_2$ side by side without any "interaction". Formally, $\mathbf{INS}_1 + \mathbf{INS}_2$ is defined as follows:

- The category of signatures of $\mathbf{INS}_1 + \mathbf{INS}_2$ is the disjoint union $\mathbf{Sign}_1 + \mathbf{Sign}_2$ of the categories of signatures of $\mathbf{INS}_1$ and of $\mathbf{INS}_2$.
- The sentence functor $\mathbf{Sen}_{\mathbf{INS}_1+\mathbf{INS}_2}: \mathbf{Sign}_1 + \mathbf{Sign}_2 \to \mathbf{Set}$ acts as $\mathbf{Sen}_1$ on $\mathbf{Sign}_1$ and as $\mathbf{Sen}_2$ on $\mathbf{Sign}_2$ (that is, $\mathbf{Sen}_{\mathbf{INS}_1+\mathbf{INS}_2}$ is determined by $\mathbf{Sen}_1$ and $\mathbf{Sen}_2$ according to the coproduct property of $\mathbf{Sign}_1 + \mathbf{Sign}_2$).
- The model functor $\mathbf{Mod}_{\mathbf{INS}_1+\mathbf{INS}_2}: (\mathbf{Sign}_1 + \mathbf{Sign}_2)^{op} \to \mathbf{Cat}$ acts as $\mathbf{Mod}_1$ on $\mathbf{Sign}_1$ and as $\mathbf{Mod}_2$ on $\mathbf{Sign}_2$ (that is, $\mathbf{Mod}_{\mathbf{INS}_1+\mathbf{INS}_2}$ is determined by $\mathbf{Mod}_1$ and $\mathbf{Mod}_2$ according to the coproduct property of $\mathbf{Sign}_1 + \mathbf{Sign}_2$).
- The family of satisfaction relations of $\mathbf{INS}_1 + \mathbf{INS}_2$ is the union of the families of satisfaction relations of $\mathbf{INS}_1$ and of $\mathbf{INS}_2$ (that is, for $\Sigma_1 \in |\mathbf{Sign}_1|$, $\models_{\mathbf{INS}_1+\mathbf{INS}_2,\Sigma_1}$ is $\models_{1,\Sigma_1}$, and for $\Sigma_2 \in |\mathbf{Sign}_2|$, $\models_{\mathbf{INS}_1+\mathbf{INS}_2,\Sigma_2}$ is $\models_{2,\Sigma_2}$). □

Example 4.1.45. Given institutions $\mathbf{INS}_1 = \langle \mathbf{Sign}_1, \mathbf{Sen}_1, \mathbf{Mod}_1, \langle \models_{1,\Sigma_1} \rangle_{\Sigma_1 \in |\mathbf{Sign}_1|} \rangle$ and $\mathbf{INS}_2 = \langle \mathbf{Sign}_2, \mathbf{Sen}_2, \mathbf{Mod}_2, \langle \models_{2,\Sigma_2} \rangle_{\Sigma_2 \in |\mathbf{Sign}_2|} \rangle$, their *product* $\mathbf{INS}_1 \times \mathbf{INS}_2$ is defined as follows:

- The category of signatures of $\mathbf{INS}_1 \times \mathbf{INS}_2$ is the product $\mathbf{Sign}_1 \times \mathbf{Sign}_2$ of the categories of signatures of $\mathbf{INS}_1$ and of $\mathbf{INS}_2$; thus a signature in $\mathbf{INS}_1 \times \mathbf{INS}_2$ is a pair consisting of one signature from $\mathbf{INS}_1$ and one from $\mathbf{INS}_2$, and similarly for signature morphisms.
- The sentence functor $\mathbf{Sen}_{\mathbf{INS}_1 \times \mathbf{INS}_2} : \mathbf{Sign}_1 \times \mathbf{Sign}_2 \to \mathbf{Set}$ is defined as follows:
  - For any signature $\langle \Sigma_1, \Sigma_2 \rangle \in |\mathbf{Sign}_1 \times \mathbf{Sign}_2|$, $\mathbf{Sen}_{\mathbf{INS}_1 \times \mathbf{INS}_2}(\langle \Sigma_1, \Sigma_2 \rangle) = \mathbf{Sen}_1(\Sigma_1) + \mathbf{Sen}_2(\Sigma_2)$ is the disjoint union of the sets of $\mathbf{INS}_1$-sentences over Σ_1 and of $\mathbf{INS}_2$-sentences over Σ_2.
  - For any signature morphism $\langle \sigma_1, \sigma_2 \rangle : \langle \Sigma_1, \Sigma_2 \rangle \to \langle \Sigma_1', \Sigma_2' \rangle$, $\mathbf{Sen}_{\mathbf{INS}_1 \times \mathbf{INS}_2}(\langle \sigma_1, \sigma_2 \rangle) = \mathbf{Sen}_1(\sigma_1) + \mathbf{Sen}_2(\sigma_2)$ acts as $\mathbf{Sen}_1(\sigma_1)$ on $\mathbf{INS}_1$-sentences and as $\mathbf{Sen}_2(\sigma_2)$ on $\mathbf{INS}_2$-sentences over the signature $\langle \Sigma_1, \Sigma_2 \rangle$.
- The model functor $\mathbf{Mod}_{\mathbf{INS}_1 \times \mathbf{INS}_2} : (\mathbf{Sign}_1 \times \mathbf{Sign}_2)^{op} \to \mathbf{Cat}$ is defined as follows:
  - For any signature $\langle \Sigma_1, \Sigma_2 \rangle \in |\mathbf{Sign}_1 \times \mathbf{Sign}_2|$, $\mathbf{Mod}_{\mathbf{INS}_1 \times \mathbf{INS}_2}(\langle \Sigma_1, \Sigma_2 \rangle) = \mathbf{Mod}_1(\Sigma_1) \times \mathbf{Mod}_2(\Sigma_2)$ is the product of the categories of $\mathbf{INS}_1$-models over Σ_1 and of $\mathbf{INS}_2$-models over Σ_2; thus a model in $\mathbf{INS}_1 \times \mathbf{INS}_2$ is a pair consisting of one model from $\mathbf{INS}_1$ and one from $\mathbf{INS}_2$, and similarly for model morphisms.
  - For any signature morphism $\langle \sigma_1, \sigma_2 \rangle : \langle \Sigma_1, \Sigma_2 \rangle \to \langle \Sigma_1', \Sigma_2' \rangle$, $\mathbf{Mod}_{\mathbf{INS}_1 \times \mathbf{INS}_2}(\langle \sigma_1, \sigma_2 \rangle) = \mathbf{Mod}_1(\sigma_1) \times \mathbf{Mod}_2(\sigma_2)$ acts as $\mathbf{Mod}_1(\sigma_1)$ on the $\mathbf{INS}_1$-components of $\langle \Sigma_1', \Sigma_2' \rangle$-models and model morphisms and as $\mathbf{Mod}_2(\sigma_2)$ on the $\mathbf{INS}_2$-components of $\langle \Sigma_1', \Sigma_2' \rangle$-models and model morphisms.
- For any signature $\langle \Sigma_1, \Sigma_2 \rangle \in |\mathbf{Sign}_1 \times \mathbf{Sign}_2|$, model $\langle M_1, M_2 \rangle \in |\mathbf{Mod}_{\mathbf{INS}_1 \times \mathbf{INS}_2}(\langle \Sigma_1, \Sigma_2 \rangle)|$ and sentences $\varphi_1 \in \mathbf{Sen}_1(\Sigma_1)$ and $\varphi_2 \in \mathbf{Sen}_2(\Sigma_2)$, $\langle M_1, M_2 \rangle \models_{\mathbf{INS}_1 \times \mathbf{INS}_2, \langle \Sigma_1, \Sigma_2 \rangle} \varphi_1$ if and only if $M_1 \models_{1, \Sigma_1} \varphi_1$, and $\langle M_1, M_2 \rangle \models_{\mathbf{INS}_1 \times \mathbf{INS}_2, \langle \Sigma_1, \Sigma_2 \rangle} \varphi_2$ if and only if $M_2 \models_{2, \Sigma_2} \varphi_2$. That is, satisfaction in $\mathbf{INS}_1 \times \mathbf{INS}_2$ is defined as $\mathbf{INS}_1$-satisfaction for $\mathbf{INS}_1$-sentences (extracting the $\mathbf{INS}_1$-components of $\mathbf{INS}_1 \times \mathbf{INS}_2$-models) and as $\mathbf{INS}_2$-satisfaction for $\mathbf{INS}_2$-sentences (extracting the $\mathbf{INS}_2$-components of $\mathbf{INS}_1 \times \mathbf{INS}_2$-models). □

The next example indicates a technically correct but intuitively somewhat artificial way of dealing with the translation of sentences along signature morphisms. The simple idea is that instead of actually translating sentences from one signature to another, we can always keep the original sentence over its original signature together with a morphism "fitting" it to another signature.

Example 4.1.46. Consider an institution $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ and a function $NewSen : |\mathbf{Sign}| \to |\mathbf{Set}|$ together with a family of relations $\langle \models_{NewSen, \Sigma} \subseteq |\mathbf{Mod}(\Sigma)| \times NewSen(\Sigma) \rangle_{\Sigma \in |\mathbf{Sign}|}$. Intuitively, for any signature Σ, $NewSen(\Sigma)$ is a set of new sentences over Σ with the satisfaction relation $\models_{NewSen, \Sigma}$. We define an institution $\mathbf{INS} + NewSen$ by adding these new sentences fitted to other signatures via signature morphisms:

- The category of signatures of $\mathbf{INS} + NewSen$ is just the category $\mathbf{Sign}$ of $\mathbf{INS}$-signatures.
- The sentence functor $\mathbf{Sen}_{\mathbf{INS} + NewSen} : \mathbf{Sign} \to \mathbf{Set}$ is defined as follows:

  - For any signature $\Sigma \in |\mathbf{Sign}|$, $\mathbf{Sen}_{\mathbf{INS}+NewSen}(\Sigma)$ is the (disjoint) union of the "old" sentences $\mathbf{Sen}(\Sigma)$ and the set[16] of "new" sentences fitted to the signature Σ by a signature morphism. The latter are pairs $\langle \varphi', \theta \rangle$, written as φ' **through** θ, with $\theta : \Sigma' \to \Sigma$ and $\varphi' \in NewSen(\Sigma')$ for an arbitrary signature Σ'.
  - For any signature morphism $\sigma : \Sigma \to \Sigma_1$, $\mathbf{Sen}_{\mathbf{INS}+NewSen}(\sigma)$ works as $\mathbf{Sen}(\sigma)$ on the **INS**-sentences; for $\theta : \Sigma' \to \Sigma$ and $\varphi' \in NewSen(\Sigma')$, $\mathbf{Sen}_{\mathbf{INS}+NewSen}(\sigma)(\varphi' \textbf{ through } \theta) = \varphi' \textbf{ through } \theta;\sigma$.

- The model functor of $\mathbf{INS} + NewSen$ is the model functor $\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ of **INS**.
- For each signature $\Sigma \in |\mathbf{Sign}|$, the Σ-satisfaction relation of $\mathbf{INS} + NewSen$ is just the same as the Σ-satisfaction relation of **INS** for the "old" Σ-sentences and then, for any Σ-model $M \in |\mathbf{Mod}(\Sigma)|$, $\theta : \Sigma' \to \Sigma$ and $\varphi' \in NewSen(\Sigma')$, $M \models_{\mathbf{INS}+NewSen} \varphi' \textbf{ through } \theta$ if and only if $M|_\theta \models_{NewSen,\Sigma'} \varphi'$.

Exercise. Check the satisfaction condition. □

We conclude this list of constructions on institutions with a sketch of how various modal (and temporal) logics may be built over an arbitrary institution.

Example 4.1.47. Let $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ be an institution. We define the institution $\mathbf{LTL}_{\mathbf{INS}}$ of linear-time temporal logic over **INS**, using sequences of models from **INS** as models and sentences from **INS** as "state sentences", that is:

- The category of signatures of $\mathbf{LTL}_{\mathbf{INS}}$ is **Sign**, the same as in **INS**.
- For each signature Σ, a Σ-model in $\mathbf{LTL}_{\mathbf{INS}}$ is a countably infinite sequence $\overline{M} = \langle M_n \rangle_{n \geq 0}$ of models $M_n \in |\mathbf{Mod}(\Sigma)|$ for $n \geq 0$. Reducts of such models w.r.t. a signature morphism σ are defined componentwise, using the reduct w.r.t. σ as defined in **INS**. (We disregard model morphisms here, taking $\mathbf{Mod}_{\mathbf{LTL}_{\mathbf{INS}}}(\Sigma)$ to be the discrete category.)
- For each signature Σ, the set of Σ-sentences in $\mathbf{LTL}_{\mathbf{INS}}$ is the least set that contains `true` and all the sentences in $\mathbf{Sen}(\Sigma)$ (called *state sentences* in this context) and is closed under negation, written $\neg\varphi$, conjunction, $\varphi \wedge \psi$, and two modal operators: *next time*, $\mathsf{X}\varphi$, and *until*, $\varphi \mathsf{U} \psi$.
- For each signature Σ, satisfaction is defined in terms of an auxiliary relation of satisfaction at a given position in the temporal sequence; for each model $\overline{M} = \langle M_n \rangle_{n \geq 0}$, and $j \geq 0$ we define:
  - for any state sentence φ, $\overline{M} \models^j \varphi$ if $M_j \models \varphi$ (in **INS**);
  - $\overline{M} \models^j \neg\varphi$ if it is not the case that $\overline{M} \models^j \varphi$;
  - $\overline{M} \models^j \varphi \wedge \psi$ if $\overline{M} \models^j \varphi$ and $\overline{M} \models^j \psi$;

[16] This may lead to some foundational difficulties, since the collection of all signature morphisms into Σ, and hence the collection of all new Σ-sentences, need not form a set. One argument for ignoring these problems here is that we can typically limit the size of the category of signatures of the institution we start with, for example assuming that the category **Sign** is small.

- $\overline{M} \models^j \mathsf{X}\varphi$ if $\overline{M} \models^{j+1} \varphi$; and
- $\overline{M} \models^j \varphi \mathsf{U} \psi$ if for some $k \geq j$, $\overline{M} \models^k \psi$ and for all $j \leq i < k$, $\overline{M} \models^i \varphi$.

We put now $\overline{M} \models_{\mathbf{LTL_{INS}},\Sigma} \varphi$ if $\overline{M} \models^0 \varphi$.

Exercise. Complete the definition and check the satisfaction condition.

Exercise. Add other temporal modalities, like "eventually/finally" and "henceforth/globally", either by defining them explicitly, or as abbreviations, for instance: $\mathsf{F}\varphi \equiv \mathsf{true}\,\mathsf{U}\,\varphi$, $\mathsf{G}\varphi \equiv \neg(\mathsf{F}(\neg\varphi))$, etc.

Also, add "past" temporal modalities (previous, since, sometimes in the past, always in the past, etc). □

Exercise 4.1.48. Proceeding similarly as in Example 4.1.47, given an institution **INS**, define the institution $\mathbf{MDL_{INS}}$ of modal logic:

- The category of signatures of $\mathbf{MDL_{INS}}$ is **Sign**, the same as in **INS**.
- For each signature Σ, a Σ-model in $\mathbf{MDL_{INS}}$ is a Kripke structure, i.e., a triple $\langle W, \rightsquigarrow, \overline{M} \rangle$, which consists of a set W (of "possible worlds" or "state names") and a relation $\rightsquigarrow \subseteq W \times W$ ("transition relation") together with a family $\overline{M} = \langle M_w \rangle_{w \in W}$ of Σ-models in **INS**, $M_w \in |\mathbf{Mod}(\Sigma)|$ for $w \in W$. Again, we disregard model morphisms.
- For each signature Σ, the set of Σ-sentences in $\mathbf{MDL_{INS}}$ is the least set that contains true and all the sentences in $\mathbf{Sen}(\Sigma)$ and is closed under negation $\neg\varphi$, conjunction $\varphi \wedge \psi$, and the modal operator $\Box\varphi$.
- For each signature Σ, satisfaction is defined in terms of an auxiliary relation of satisfaction at a given world in a Kripke structure; here is the crucial clause:
  - $\langle W, \rightsquigarrow, \overline{M} \rangle \models^w \Box\varphi$ if for all $v \in W$ such that $w \rightsquigarrow v$, $\langle W, \rightsquigarrow, \overline{M} \rangle \models^v \varphi$.

  Then a model satisfies a sentence in $\mathbf{MDL_{INS}}$ if the sentence holds in the above sense at each of its possible worlds.

Complete the definition and check the satisfaction condition.

To keep the definition closer to $\mathbf{LTL_{INS}}$, you may want to define a somewhat different version of modal logic, where Kripke structures in addition indicate an initial world, and then the satisfaction of a sentence in a model is determined by its satisfaction at this initial world. You may also want to impose requirements on the transition relation (for instance, that it is transitive, or that all possible worlds can be reached from the initial world, etc.).

Combining the ideas behind $\mathbf{MDL_{INS}}$ and $\mathbf{LTL_{INS}}$, define the institution $\mathbf{CTL^*_{INS}}$ of branching-time temporal logic, where signatures and models are as in $\mathbf{MDL_{INS}}$, but sentences are closed under a variety of temporal operators used to quantify (separately) over paths in the Kripke structure and over worlds in these paths. HINT: Distinguish two kinds of sentences: path sentences that are evaluated for a given path in the Kripke structure; and state sentences that are evaluated for a given world in the Kripke structure — or see [Eme90].

You may also start by defining a simpler institution $\mathbf{CTL_{INS}}$ where the use of temporal operators is limited by requiring that quantification over paths and over

worlds in these paths always happen together, so in fact we have only bundled path/state temporal operators, like "for some path, always in this path", "for some path, eventually in this path", etc. □

Exercise 4.1.49. Consider an institution $\mathbf{MDL}_{\mathbf{FOPEQ}}$ of modal logic built over first-order predicate logic with equality. Note that this is *not* the institution of first-order modal logic, since quantification is internal to state sentences only and cannot be interleaved with the modal operator. Define an institution of first-order modal logic in which such an arbitrary interleaving of quantifiers, propositional connectives and the modal operator is allowed. HINT: The trouble here is with moving valuations of variables from one world to another in the definition of satisfaction. At least, define such an institution assuming that the carriers of all models in any Kripke structure coincide. Discuss possible generalisations.

Carry out similar constructions of first-order temporal logics that extend $\mathbf{LTL}_{\mathbf{FOPEQ}}$, $\mathbf{CTL}^{*}_{\mathbf{FOPEQ}}$ and $\mathbf{CTL}_{\mathbf{FOPEQ}}$, respectively. □

4.2 Flat specifications in an arbitrary institution

Throughout this section we will deal with an arbitrary but fixed institution. This means that we will be working with a logical system about which we know nothing beyond what is given in the definition of an institution. For example, we will not be able to refer to any particular components of signatures, any particular syntax of sentences, any particular internal structure of models, or any particular definition of satisfaction. Indeed, we cannot even be sure that signatures have components, that sentences are syntactic entities in any sense, or that models have any internal structure at all.

Given these limitations, one may think that there is very little that can be done. However, the structure of an institution is rich enough to allow us to recast in these terms the material on simple equational specifications presented in Sections 2.2 and 2.3 (this will be done in the present section, without repeating the discussion and motivation) and then to proceed further into the theory of specifications and software development.

Let us then fix an arbitrary institution $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$. We start with the basic concepts built around the notion of satisfaction.

Definition 4.2.1 ($Mod_\Sigma(\Phi)$, $Th_\Sigma(\mathscr{M})$, $Cl_\Sigma(\Phi)$ and $Cl_\Sigma(\mathscr{M})$). Let Σ be an arbitrary signature.

- For any set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences, the class $Mod_\Sigma(\Phi) \subseteq |\mathbf{Mod}(\Sigma)|$ of *models of* Φ is defined as the class of all Σ-models that satisfy all the sentences in Φ.[17]

[17] Note the overloading of the term "model" as discussed after Definition 4.1.1. We continue to follow the terminology of [GB92], hoping that this will not lead to any confusion.

- For any class $\mathscr{M} \subseteq |\mathbf{Mod}(\Sigma)|$ of Σ-models, the *theory of* $\mathscr{M}$ is the set $Th_\Sigma(\mathscr{M}) \subseteq \mathbf{Sen}(\Sigma)$ of all the Σ-sentences that are satisfied by all the models in $\mathscr{M}$.
- A set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences is *closed* if $\Phi = Th_\Sigma(Mod_\Sigma(\Phi))$. We will write $Cl_\Sigma(\Phi)$ for $Th_\Sigma(Mod_\Sigma(\Phi))$ and refer to $Cl_\Sigma(\Phi)$ as the *closure of* Φ.
- A class $\mathscr{M} \subseteq |\mathbf{Mod}(\Sigma)|$ of Σ-models is *closed* if $\mathscr{M} = Mod_\Sigma(Th_\Sigma(\mathscr{M}))$. Closed classes of models will be called *definable*. The *closure* of $\mathscr{M}$ is the class $Mod_\Sigma(Th_\Sigma(\mathscr{M}))$. □

The basic properties of the above notions follow from the fact that Th_Σ and Mod_Σ form a Galois connection:

Proposition 4.2.2. *For any signature Σ, the mappings Th_Σ and Mod_Σ form a Galois connection between sets of Σ-sentences and classes of Σ-models ordered by inclusion.*

Proof. The proof is just the same (and just as easy) as in the equational case, cf. Proposition 2.3.2. □

Corollary 4.2.3. *For any signature Σ, set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences, and class $\mathscr{M} \subseteq |\mathbf{Mod}(\Sigma)|$ of Σ-models:*

$$\Phi \subseteq Th_\Sigma(\mathscr{M}) \quad \textit{iff} \quad Mod_\Sigma(\Phi) \supseteq \mathscr{M}$$ □

Exercise 4.2.4. Construct counterexamples that show that under the assumptions of Corollary 4.2.3 neither of the following implications holds:

$$Mod_\Sigma(\Phi) \subseteq \mathscr{M} \text{ implies } Th_\Sigma(\mathscr{M}) \subseteq \Phi$$
$$Th_\Sigma(\mathscr{M}) \subseteq \Phi \text{ implies } Mod_\Sigma(\Phi) \subseteq \mathscr{M}.$$

Prove that the former implication holds if Φ is closed, and the latter if $\mathscr{M}$ is closed (i.e., is definable). □

The satisfaction relation determines in the obvious way a consequence relation between sentences of the institution:

Definition 4.2.5 (Semantic consequence). Let Σ be an arbitrary signature. A Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$ is a *semantic consequence* of a set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences, written $\Phi \models_\Sigma \varphi$, if $\varphi \in Cl_\Sigma(\Phi)$, or equivalently, if $Mod_\Sigma(\Phi) \models_\Sigma \varphi$. □

As usual, the subscript Σ will often be omitted.

In the following we will often implicitly rely on three basic properties of semantic consequence, namely that it is reflexive, closed under weakening, and transitive, in the following sense:

Proposition 4.2.6. *Let Σ be a signature. Consider any Σ-sentences $\varphi, \psi \in \mathbf{Sen}(\Sigma)$, and sets of Σ-sentences $\Phi, \Psi \subseteq \mathbf{Sen}(\Sigma)$, and $\Psi_\varphi \subseteq \mathbf{Sen}(\Sigma)$ for each $\varphi \in \Phi$. Then:*

1. $\{\varphi\} \models_\Sigma \varphi$.
2. *If $\Phi \models_\Sigma \varphi$ then $\Phi \cup \Psi \models_\Sigma \varphi$.*

3. If $\Phi \models_\Sigma \psi$ *and* $\Psi_\varphi \models_\Sigma \varphi$ *for each* $\varphi \in \Phi$ *then* $\bigcup_{\varphi\in\Phi} \Psi_\varphi \models_\Sigma \psi$.

Proof. Directly from the definition. □

Another important property of semantic consequence is that it is preserved by translation along signature morphisms:

Proposition 4.2.7. *For any signature morphism* $\sigma: \Sigma \to \Sigma'$, *set* $\Phi \subseteq \mathbf{Sen}(\Sigma)$ *of* Σ*-sentences, and* Σ*-sentence* $\varphi \in \mathbf{Sen}(\Sigma)$,

$$\Phi \models_\Sigma \varphi \;\; \textit{implies} \;\; \sigma(\Phi) \models_{\Sigma'} \sigma(\varphi).$$

In other words, $\sigma(Cl_\Sigma(\Phi)) \subseteq Cl_{\Sigma'}(\sigma(\Phi))$.

Proof. Let $M' \in Mod_{\Sigma'}(\sigma(\Phi))$. Then by the satisfaction condition $M'\big|_\sigma \in Mod_\Sigma(\Phi)$, and so by the definition of the consequence relation $M'\big|_\sigma \models \varphi$. Thus, by the satisfaction condition again, $M' \models \sigma(\varphi)$, which shows that indeed $\sigma(\Phi) \models \sigma(\varphi)$. □

In general, the reverse implication does not hold, that is, the consequence relation is not reflected by translation along signature morphisms.

Exercise 4.2.8. Try to prove the opposite implication, and notice where the proof breaks down. Then construct a counterexample showing that $\sigma(\Phi) \models \sigma(\varphi)$ does not imply that $\Phi \models \varphi$ even in the standard equational institution **EQ**. (HINT: See Proposition 4.2.15 below.) □

Corollary 4.2.9. *Under the assumptions of Proposition 4.2.7,* $Cl_{\Sigma'}(\sigma(Cl_\Sigma(\Phi))) = Cl_{\Sigma'}(\sigma(\Phi))$. □

The above corollary implies that when we want to "move" the closure of a set of sentences from one signature to another, it is enough to move only the set itself; all its consequences can be derived over the target signature as well.

Another consequence of Proposition 4.2.7 is that closure of a set of sentences is reflected by translation along signature morphisms:

Corollary 4.2.10. *For any signature morphism* $\sigma: \Sigma \to \Sigma'$ *and set* $\Phi' \subseteq \mathbf{Sen}(\Sigma')$ *of* Σ'*-sentences, if* Φ' *is closed then so is* $\sigma^{-1}(\Phi')$.

Proof. Suppose Φ' is closed and let $\varphi \in Cl_\Sigma(\sigma^{-1}(\Phi'))$. First, notice that since $\sigma(\sigma^{-1}(\Phi')) \subseteq \Phi'$, $Cl_{\Sigma'}(\sigma(\sigma^{-1}(\Phi'))) \subseteq Cl_{\Sigma'}(\Phi')$. Now, by Proposition 4.2.7, $\sigma(\varphi) \in Cl_{\Sigma'}(\sigma(\sigma^{-1}(\Phi'))) \subseteq Cl_{\Sigma'}(\Phi') = \Phi'$. Thus, $\varphi \in \sigma^{-1}(\Phi')$. □

Notice that the above does not imply that "closure commutes with inverse image" in general; only one of the required inclusions holds:

Corollary 4.2.11. *For any signature morphism* $\sigma: \Sigma \to \Sigma'$, *set* $\Phi' \subseteq \mathbf{Sen}(\Sigma')$ *of* Σ'*-sentences, and* Σ*-sentence* $\varphi \in \mathbf{Sen}(\Sigma)$, *if* $\sigma^{-1}(\Phi') \models \varphi$ *then* $\Phi' \models \sigma(\varphi)$. *In other words,* $Cl_\Sigma(\sigma^{-1}(\Phi')) \subseteq \sigma^{-1}(Cl_{\Sigma'}(\Phi'))$. □

Exercise 4.2.12. Show that the reverse inclusion does not hold in the standard equational institution **EQ**. □

Forming the closure of a set of sentences consists of two phases: first taking the class of models the set defines, and then taking the theory of this class. Separation of these two phases by translation along a signature morphism preserves the closure to some extent only:

Proposition 4.2.13. *For any signature morphism $\sigma: \Sigma \to \Sigma'$ and set $\Phi' \subseteq \mathbf{Sen}(\Sigma')$ of Σ'-sentences,*

$$Cl_\Sigma(\sigma^{-1}(\Phi')) \subseteq Th_\Sigma(Mod_{\Sigma'}(\Phi')|_\sigma) = \sigma^{-1}(Cl_{\Sigma'}(\Phi'))$$

where for any class $\mathscr{M} \subseteq |\mathbf{Mod}(\Sigma')|$, $\mathscr{M}|_\sigma = \{M'|_\sigma \mid M' \in \mathscr{M}\}$.

Proof. For the first part, let $\varphi \in Cl_\Sigma(\sigma^{-1}(\Phi'))$. Then, by Corollary 4.2.11, $\Phi' \models_{\Sigma'} \sigma(\varphi)$. Hence, by the satisfaction condition, $Mod_{\Sigma'}(\Phi')|_\sigma \models_\Sigma \varphi$, and so $\varphi \in Th_\Sigma(Mod_{\Sigma'}(\Phi')|_\sigma)$.

Since $Mod_{\Sigma'}(\Phi') = Mod_{\Sigma'}(Cl_{\Sigma'}(\Phi'))$, this shows $Th_\Sigma(Mod_{\Sigma'}(\Phi')|_\sigma) = Th_\Sigma(Mod_{\Sigma'}(Cl_{\Sigma'}(\Phi'))|_\sigma) \supseteq Cl_\Sigma(\sigma^{-1}(Cl_{\Sigma'}(\Phi'))) \supseteq \sigma^{-1}(Cl_{\Sigma'}(\Phi'))$, and hence also proves one inclusion ("$\supseteq$") of the second part. For the opposite inclusion, consider $\varphi \in Th_\Sigma(Mod_{\Sigma'}(\Phi')|_\sigma)$, that is $Mod_{\Sigma'}(\Phi')|_\sigma \models_\Sigma \varphi$. By the satisfaction condition, $Mod_{\Sigma'}(\Phi') \models_{\Sigma'} \sigma(\varphi)$, which means $\sigma(\varphi) \in Cl_{\Sigma'}(\Phi')$, and so indeed $\varphi \in \sigma^{-1}(Cl_{\Sigma'}(\Phi'))$. □

Corollary 4.2.14. *For any signature morphism $\sigma: \Sigma \to \Sigma'$ and set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences, $Cl_\Sigma(\Phi) \subseteq \sigma^{-1}(Cl_{\Sigma'}(\sigma(\Phi)))$.* □

Just as the implication opposite to the one stated in Proposition 4.2.7 does not hold in general, the inclusion opposite to the one above does not hold in general either. This changes for *surjective* reduct functors.

Proposition 4.2.15. *Consider a signature morphism $\sigma: \Sigma \to \Sigma'$ such that the reduct functor $\_|_\sigma: \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ is surjective on models. For any set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences and Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$,*

$$\Phi \models_\Sigma \varphi \quad \textit{iff} \quad \sigma(\Phi) \models_{\Sigma'} \sigma(\varphi).$$

Proof. We prove only the implication opposite to that of Proposition 4.2.7. Let $M \in |\mathbf{Mod}(\Sigma)|$ be an arbitrary Σ-model, and let $M' \in |\mathbf{Mod}(\Sigma')|$ be a σ-expansion of M, i.e., $M'|_\sigma = M$ (such an M' exists since $\_|_\sigma$ is surjective on models). If $M \models_\Sigma \Phi$ then by the satisfaction condition $M' \models_{\Sigma'} \sigma(\Phi)$, and so $M' \models_{\Sigma'} \sigma(\varphi)$. Thus, by the satisfaction condition again, $M \models_\Sigma \varphi$. □

Corollary 4.2.16. *Under the assumptions of Proposition 4.2.15, $Cl_\Sigma(\Phi) = \sigma^{-1}(Cl_{\Sigma'}(\sigma(\Phi)))$.* □

This shows that the surjectivity of the reduct functor ensures that moving along a signature morphism is "sound" and "complete" as a strategy for deciding if $\Phi \models_\Sigma \varphi$ by checking whether or not $\sigma(\Phi) \models_{\Sigma'} \sigma(\varphi)$ — without this property, such a strategy is still "complete" (the satisfaction condition ensures that no consequences are lost) but is not always "sound" (new consequences between "old" sentences may be added).

Exercise 4.2.17. Provide an example showing that surjectivity of $\_|_\sigma : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ is not a necessary condition for the conclusions of Proposition 4.2.15 and Corollary 4.2.16. □

Exercise 4.2.18. Show that the inclusion $Cl_\Sigma(\Phi) \subseteq \sigma^{-1}(Cl_{\Sigma'}(\sigma(\Phi)))$, for any $\sigma : \Sigma \to \Sigma'$ and $\Phi \subseteq \mathbf{Sen}(\Sigma)$, directly implies (and, in fact, is equivalent to) Corollary 4.2.11. However, the opposite inclusion $Cl_\Sigma(\Phi) \supseteq \sigma^{-1}(Cl_{\Sigma'}(\sigma(\Phi)))$ does not imply the opposite to the inclusion there: even under the assumptions of Proposition 4.2.15 and Corollary 4.2.16, the inclusion $Cl_\Sigma(\sigma^{-1}(\Phi')) \supseteq \sigma^{-1}(Cl_{\Sigma'}(\Phi'))$ may fail for a set $\Phi' \subseteq \mathbf{Sen}(\Sigma')$ of Σ'-sentences. (HINT: One way to construct a counterexample is to add *false* to the set of sentences of **EQ** for some, but not all signatures.)

Show, however, that under the assumptions of Proposition 4.2.15, for any set $\Phi' \subseteq \mathbf{Sen}(\Sigma')$ of Σ'-sentences, $Cl_\Sigma(\sigma^{-1}(\Phi')) = Th_\Sigma(Mod_{\Sigma'}(\Phi')|_\sigma)$ and $Cl_\Sigma(\sigma^{-1}(\Phi')) = \sigma^{-1}(Cl_{\Sigma'}(\Phi'))$ provided that in addition $\sigma : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$ is surjective. Discuss why this fact does not seem very interesting. □

The following generalisation of Proposition 4.2.15 underlies the key corollary below.

Proposition 4.2.19. *Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism. Suppose that a set $\Gamma \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences exactly characterises the σ-reducts of Σ'-models that satisfy a set $\Gamma' \subseteq \mathbf{Sen}(\Sigma')$ of Σ'-sentences, that is, $Mod_\Sigma(\Gamma) = \mathbf{Mod}(\sigma)(Mod_{\Sigma'}(\Gamma'))$. Then for any set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences and Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$, $\Phi \cup \Gamma \models_\Sigma \varphi$ if and only if $\sigma(\Phi) \cup \Gamma' \models_{\Sigma'} \sigma(\varphi)$.*

Proof. For the "if" part, assume that $\sigma(\Phi) \cup \Gamma' \models_{\Sigma'} \sigma(\varphi)$ and let $M \models_\Sigma \Phi \cup \Gamma$. Then, since $M \in Mod_\Sigma(\Gamma)$, there exists $M' \in Mod_{\Sigma'}(\Gamma')$ with $M'|_\sigma = M$. By the satisfaction condition, $M' \models_{\Sigma'} \sigma(\Phi)$, hence $M' \models_{\Sigma'} \sigma(\Phi) \cup \Gamma'$ and so $M' \models_{\Sigma'} \sigma(\varphi)$ as well. Thus, by the satisfaction condition again, $M \models_\Sigma \varphi$.

For the "only if" part, asume that $\Phi \cup \Gamma \models_\Sigma \varphi$ and let $M' \models_{\Sigma'} \sigma(\Phi) \cup \Gamma'$. Then by the satisfaction condition, $M'|_\sigma \models_\Sigma \Phi$ and moreover, by the assumption, $M'|_\sigma \models_\Sigma \Gamma$. Hence, $M'|_\sigma \models_\Sigma \Phi \cup \Gamma$, and so $M'|_\sigma \models_\Sigma \varphi$ as well, which by the satisfaction condition again proves that $M' \models_{\Sigma'} \sigma(\varphi)$. □

Corollary 4.2.20. *Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism. Suppose that a set $\Gamma \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences exactly characterises the σ-reducts of Σ'-models, that is, $Mod_\Sigma(\Gamma) = \mathbf{Mod}(\sigma)(|\mathbf{Mod}(\Sigma')|)$. Then for any set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences and Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$, $\Phi \cup \Gamma \models_\Sigma \varphi$ if and only if $\sigma(\Phi) \models_{\Sigma'} \sigma(\varphi)$.* □

Exercise 4.2.21. Show that Proposition 4.2.15 follows directly from Proposition 4.2.19 (or Corollary 4.2.20). Generalise Corollary 4.2.16 in a similar way. □

Definition 4.2.22 (Presentation). For any signature Σ, a Σ-*presentation* (also known as a *flat specification*) is a pair $\langle \Sigma, \Phi \rangle$ where $\Phi \subseteq \mathbf{Sen}(\Sigma)$. $M \in |\mathbf{Mod}(\Sigma)|$ is a *model* of a Σ-presentation $\langle \Sigma, \Phi \rangle$ if $M \models \Phi$. $Mod[\langle \Sigma, \Phi \rangle]$ denotes the class of all models of the presentation $\langle \Sigma, \Phi \rangle$, and $\mathbf{Mod}[\langle \Sigma, \Phi \rangle]$ the full subcategory of $\mathbf{Mod}(\Sigma)$ with objects in $Mod[\langle \Sigma, \Phi \rangle]$. □

Definition 4.2.23 (The category of theories). For any signature Σ, a *Σ-theory* T is a Σ-presentation $\langle \Sigma, \Phi \rangle$ where Φ is closed. A Σ-presentation $\langle \Sigma, \Psi \rangle$ *presents* the Σ-theory $\langle \Sigma, Cl_\Sigma(\Psi) \rangle$.

For any theories $T = \langle \Sigma, \Phi \rangle$ and $T' = \langle \Sigma', \Phi' \rangle$, a *theory morphism* $\sigma: T \to T'$ is a signature morphism $\sigma: \Sigma \to \Sigma'$ such that $\sigma(\varphi) \in \Phi'$ for every $\varphi \in \Phi$.

The category $\mathbf{Th}_{\mathbf{INS}}$ of theories in **INS** has theories as objects and theory morphisms as morphisms, with identities and composition inherited from the category $\mathbf{Sign}_{\mathbf{INS}}$ of signatures of **INS**. □

The satisfaction condition implies the following important characterisation of theory morphisms, analogous to that given for equational theory morphisms in Proposition 2.3.13.

Proposition 4.2.24. *For any signature morphism $\sigma: \Sigma \to \Sigma'$ and sets $\Phi \subseteq \mathbf{Sen}(\Sigma)$ and $\Phi' \subseteq \mathbf{Sen}(\Sigma')$ of sentences, the following conditions are equivalent:*

1. *σ is a theory morphism $\sigma: \langle \Sigma, Cl_\Sigma(\Phi) \rangle \to \langle \Sigma', Cl_{\Sigma'}(\Phi') \rangle$.*
2. *$\sigma(\Phi) \subseteq Cl_{\Sigma'}(\Phi')$.*
3. *For every $M' \in Mod_{\Sigma'}(\Phi')$, $M'|_\sigma \in Mod_\Sigma(\Phi)$.*

Proof.

$1 \Rightarrow 2$: Obvious, since $\Phi \subseteq Cl_\Sigma(\Phi)$.

$2 \Rightarrow 3$: Consider $M' \in Mod_{\Sigma'}(\Phi')$. Then also $M' \in Mod_{\Sigma'}(Cl_{\Sigma'}(\Phi'))$, and so for all $\varphi \in \Phi$, $M' \models \sigma(\varphi)$ (since $\sigma(\varphi) \in Cl_{\Sigma'}(\Phi')$). Hence, by the satisfaction condition, $M'|_\sigma \models \varphi$, and thus indeed $M'|_\sigma \in Mod_\Sigma(\Phi)$.

$3 \Rightarrow 1$: Consider any $\varphi \in Cl_\Sigma(\Phi)$. We have to show that $\sigma(\varphi) \in Cl_{\Sigma'}(\Phi')$, that is that for all $M' \in Mod_{\Sigma'}(\Phi')$, $M' \models \sigma(\varphi)$. However, if $M' \in Mod_{\Sigma'}(\Phi')$ then $M'|_\sigma \in Mod_\Sigma(\Phi)$. Hence, $M'|_\sigma \models \varphi$, and the conclusion follows from the satisfaction condition. □

Exercise 4.2.25. Define the category $\mathbf{Pres}_{\mathbf{INS}}$ of presentations in **INS**, with morphisms $\sigma: \langle \Sigma, \Phi \rangle \to \langle \Sigma', \Phi' \rangle$ that are signature morphisms $\sigma: \Sigma \to \Sigma'$ such that $\Phi' \models \sigma(\varphi)$ for all $\varphi \in \Phi$. Check that $\mathbf{Th}_{\mathbf{INS}}$ is a full subcategory of $\mathbf{Pres}_{\mathbf{INS}}$ and that the two categories are equivalent. □

Exercise 4.2.26. Show that by Proposition 4.2.24 above, the mapping which to any theory assigns the category of its models extends to a functor $\mathbf{Mod} : \mathbf{Th}_{\mathbf{INS}}^{op} \to \mathbf{Cat}$, where:

- for any theory $T = \langle \Sigma, \Phi \rangle$, $\mathbf{Mod}[T]$ is the full subcategory of $\mathbf{Mod}(\Sigma)$ with objects in $Mod[T]$ as in Definition 4.2.22; and
- for any theory morphism $\sigma: T \to T'$, $\mathbf{Mod}(\sigma)$ is the reduct functor $\_|_\sigma : \mathbf{Mod}[T'] \to \mathbf{Mod}[T]$. □

Many standard properties of theories (and presentations) investigated in the realm of classical model theory may be formulated in the framework of an arbitrary institution. For example:

Definition 4.2.27 (Consistency and completeness of a presentation). A presentation $\langle \Sigma, \Phi \rangle$ is *consistent* if it has a model, i.e. if $Mod[\langle \Sigma, \Phi \rangle] \neq \emptyset$.

A presentation $\langle \Sigma, \Phi \rangle$ is *complete* if it is a maximal consistent presentation, i.e. if it is consistent and no presentation $\langle \Sigma, \Phi' \rangle$ such that Φ' properly contains Φ is consistent. □

Proposition 4.2.28. *A presentation $\langle \Sigma, \Phi \rangle$ is consistent if and only if the theory $\langle \Sigma, Cl_\Sigma(\Phi) \rangle$ is consistent. Any complete presentation is a (consistent) theory.* □

Definition 4.2.29 (Conservative theory morphism). For any theories $T = \langle \Sigma, \Phi \rangle$ and $T' = \langle \Sigma', \Phi' \rangle$, a theory morphism $\sigma: T \to T'$ is *conservative* if for every Σ-sentence φ, $\varphi \in \Phi$ whenever $\sigma(\varphi) \in \Phi'$.

A theory morphism $\sigma: T \to T'$ *admits model expansion* if the corresponding reduct function $-|_\sigma : Mod_{\Sigma'}(\Phi') \to Mod_\Sigma(\Phi)$ is surjective, that is, for every Σ-model M such that $M \models_\Sigma \Phi$, there exists a Σ'-model M' such that $M' \models_{\Sigma'} \Phi'$ and $M'|_\sigma = M$. □

Exercise 4.2.30. As in Proposition 4.2.15, show that a theory morphism $\sigma: T \to T'$ is conservative if it admits model expansion. Note that the opposite implication does not hold by Exercise 4.2.17. □

The careful reader has probably realised that in this section we have not even mentioned model morphisms. Indeed, everything above works equally well if we forget about the category structure provided on the collections of models in an institution. But this proves inadequate for some purposes; see for example the next section where the category structure on models is exploited.

4.3 Constraints

As discussed in Section 2.5, the class of all models that satisfy a given presentation often contains some models that intuitively are undesirable realisations of the presentation. Different methods are used to constrain the semantics of presentations so that from among all its models only the ones that are "desirable" are selected: for example, one may take its initial semantics, reachable semantics, final semantics, etc. (cf. Sections 2.5 and 2.7.2). How do these fit into the institutional framework introduced above? Let us consider initiality constraints[18] first.

There is clearly no problem with expressing the basic concept of initial model in an arbitrary institution: models over any signature form a category, hence the class of models satisfying a given presentation determines a full subcategory of this category — and we know what initiality means in any category (cf. Section 3.2.1).

Let $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ be an institution, fixed throughout this section.

[18] We use the term "constraint" here following the terminology of [BG80], [GB92]. Initiality and data constraints as discussed and formally defined below have nothing to do with constraints as used in "constraint logic programming" [JL87].

Definition 4.3.1 (Initial model of a presentation). For any signature $\Sigma \in |\mathbf{Sign}|$ and set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of sentences, the *initial model* of the presentation $\langle \Sigma, \Phi \rangle$ is the (unique up to isomorphism) initial object in $Mod_\Sigma(\Phi)$ considered as a full subcategory of $\mathbf{Mod}(\Sigma)$. □

We might feel tempted to pursue a number of possibilities to incorporate the idea of initiality into the institutional framework:

- We may hope to be able to modify all institutions of interest so that they yield initial semantics directly, by changing the model functor **Mod** to yield only the initial models as models over any signature. Clearly, this fails: requiring initiality only makes sense relative to a presentation. If sentences are not taken into account then for example the only initial models in the institution **EQ** of equational logic are ground term algebras.
- We can attempt to modify the satisfaction relation so that only the initial models of a sentence will be defined to satisfy it. Quite obviously, this does not work, since it would then be impossible to adequately define models of presentations involving more than one sentence. Without modifying the satisfaction relation, we could modify Definitions 4.2.1 and 4.2.22 and consider only initial models of presentations by defining $Mod_\Sigma(\Phi)$ to consist only of the initial models in $\{M \mid M \models \Phi\}$ considered as a full subcategory of $\mathbf{Mod}(\Sigma)$. But this would make the whole theory rather clumsy, and the various definitions would not fit together as neatly as they do now. For example, Propositions 4.2.7 and 4.2.24 would no longer hold. Worse, this would not allow the user to write axioms that are to be interpreted in a loose, non-initial fashion, indicating that only certain parts of a specification are to be interpreted in an initial way. See Example 4.3.2 below.
- We can view the requirement of initiality with respect to a presentation as just another *sentence*. This would be a rather complicated sentence, as it has to contain other sentences within it, but in view of examples like 4.1.38 (not to mention 4.1.35) there is no reason why this should bother us. This is the approach we will take.

It is not sufficient to define initiality constraints simply as sets of sentences over a given signature, and then to define their satisfaction via the notion of an initial model. The problem is that we do not always want to constrain the entire model of a presentation. As the following example illustrates, we need to be able to constrain only a certain part of this model, that is, to impose initiality constraints on its reduct to a certain subsignature.

Example 4.3.2. Recall Exercise 2.5.21 which concerned the specification of a function $ch\colon nat \to nat$ that for each natural number n chooses an arbitrary number that is greater than n. As argued there, we certainly do not want to take the initial model of the entire specification: the initial model would generate "artificial elements" of sort *nat* (as the results of the function *ch*) and then artificial elements of sort *bool* as well (as results of comparisons by $<$ involving the artificial elements of sort *nat*). What one would like is to first interpret the original specification NAT of natural numbers in an initial way, do the same for the specification BOOL, add the operation

$\_\_ < \_\_ : nat \times nat \to bool$ (which is defined by its axioms in a sufficiently complete way) — it so happens that this would be the same as taking an initial model of these specifications put together — and only then add an operation $ch : nat \to nat$ with the corresponding axiom interpreted in the underlying logic, with no initiality restrictions intervening in any way at this stage. □

By allowing initiality requirements to be "fitted" to larger signatures by signature morphisms, along the lines of the construction presented in Example 4.1.46, we can impose the initiality requirement on parts of models.

Definition 4.3.3 (Initiality constraint). Let $\Sigma \in |\mathbf{Sign}|$ be a signature. A Σ-*initiality constraint* is a pair $\langle \Phi', \theta \rangle$, written as **initial** Φ' **through** θ, where $\theta : \Sigma' \to \Sigma$ is a signature morphism and $\Phi' \subseteq \mathbf{Sen}(\Sigma')$ is a set of Σ'-sentences. A Σ-model $M \in |\mathbf{Mod}(\Sigma)|$ *satisfies* a Σ-initiality constraint **initial** Φ' **through** θ if its reduct $M|_\theta \in |\mathbf{Mod}(\Sigma')|$ is an initial model of $\langle \Sigma', \Phi' \rangle$. □

Now, such an initiality constraint may be regarded as just another sentence in a presentation, and freely mixed with "ordinary" sentences.

Exercise 4.3.4. Redo Exercise 2.5.21 using initiality constraints. Discuss the possibility of achieving the same effect without the "fitting morphism" component in initiality constraints. □

The specification built in Exercise 4.3.4 is not a presentation in **FOEQ** — we have to extend this institution by adding initiality constraints first. Indeed, given an institution **INS** we can always form a new institution $\mathbf{INS}^{init}$ in which initiality constraints are allowed as additional sentences. Such a construction is implicitly involved whenever initiality constraints are used.

Definition 4.3.5 (Institution with initiality constraints). The institution $\mathbf{INS}^{init}$ with initiality constraints in **INS** is defined as follows:

- The category $\mathbf{Sign}_{\mathbf{INS}^{init}}$ of signatures is just **Sign**, the same as in **INS**.
- The functor $\mathbf{Sen}_{\mathbf{INS}^{init}}$ gives:
  - for each signature Σ, the (disjoint) union of the set $\mathbf{Sen}(\Sigma)$ of Σ-sentences in **INS** and of the set of Σ-initiality constraints;[19] and
  - for each signature morphism $\sigma : \Sigma \to \Sigma_1$, the translation function $\mathbf{Sen}_{\mathbf{INS}^{init}}(\sigma)$ that works as $\mathbf{Sen}(\sigma)$ on all the "old" Σ-sentences in **INS**, and for any Σ-initiality constraint **initial** Φ' **through** θ, where $\theta : \Sigma' \to \Sigma$ and $\Phi' \subseteq \mathbf{Sen}(\Sigma')$, is defined by $\mathbf{Sen}_{\mathbf{INS}^{init}}(\sigma)(\textbf{initial } \Phi' \textbf{ through } \theta) = \textbf{initial } \Phi' \textbf{ through } \theta;\sigma$.
- The functor $\mathbf{Mod}_{\mathbf{INS}^{init}}$ is just **Mod**, the same as in **INS**.
- For each signature $\Sigma \in |\mathbf{Sign}_{\mathbf{INS}^{init}}|$, the Σ-satisfaction relation $\models_{\mathbf{INS}^{init}, \Sigma}$ is the same as the Σ-satisfaction relation in **INS** for the Σ-sentences from **INS**, and is given by Definition 4.3.3 for Σ-initiality constraints. □

[19] As in Example 4.1.46, this may lead to some foundational difficulties which we disregard here, cf. footnote 16.

Exercise 4.3.6. Present the above definition as an instance of the construction given in Example 4.1.46. Notice that this is sufficient to conclude that $\mathbf{INS}^{init}$ is indeed an institution.

Show (referring for example to Exercise 4.3.4) that in general the translation of an initiality constraint cannot be given without the "fitting morphism" component, and so we would not be able to define an institution where only initiality constraints with trivial (identity) fitting morphisms would be allowed. □

Exercise 4.3.7. Working in the institution **EQ**, follow Definition 4.3.3 and define *reachability constraints* that are satisfied only by algebras having an indicated reduct that is reachable. Note that axioms used in initiality constraints play no role here, so you can adopt a syntax like **reachable through** θ. Following Definition 4.3.5, define an institution $\mathbf{EQ}^{reach}$ extending **EQ** by reachability constraints.

Assuming that each category of models in **INS** comes equipped with a factorisation system (Section 3.3), introduce reachability constraints for **INS** using Definition 3.3.7 and extend **INS** correspondingly. □

The use of initiality constraints as introduced above is not always entirely satisfactory. Often, rather than requiring that a certain part of a model is initial, we want to require it to be a *free extension* of some other part. Natural examples arise when we want to specify data structures built on an arbitrary set of elements, like lists, sets or bags of arbitrary elements. This involves imposing the requirement that an algebra modelling the data structure is a free extension of its reduct to the sort of elements. To formalise this, the concept of a data constraint is introduced below.

Definition 4.3.8 (Data constraint). Let $\Sigma \in |\mathbf{Sign}|$ be a signature.

A Σ*-data constraint* is a triple $\langle \sigma, \Phi', \theta \rangle$, written as **data** Φ' **over** σ **through** θ, where $\sigma : \Sigma_1 \to \Sigma'$ and $\theta : \Sigma' \to \Sigma$ are signature morphisms and $\Phi' \subseteq \mathbf{Sen}(\Sigma')$ is a set of Σ'-sentences.

A Σ-model $M \in |\mathbf{Mod}(\Sigma)|$ *satisfies* the data constraint **data** Φ' **over** σ **through** θ if its reduct $M|_\theta \in |\mathbf{Mod}(\Sigma')|$ to a Σ'-model is a free model of Φ' w.r.t. the reduct functor $-|_\sigma : \mathbf{Mod}[\langle \Sigma, \Phi' \rangle] \to \mathbf{Mod}(\Sigma_1)$ over $(M|_\theta)|_\sigma$, with the identity as unit. That is, M satisfies **data** Φ' **over** σ **through** θ if:

- $M|_\theta \models_{\Sigma'} \Phi'$; and
- for any $M' \in Mod_{\Sigma'}(\Phi')$ and Σ_1-morphism $f : M|_{\sigma;\theta} \to M'|_\sigma$ there exists a unique Σ'-morphism $f^\# : M|_\theta \to M'$ such that $f^\#|_\sigma = f$. □

Exercise 4.3.9. Using data constraints, give a specification of finite bags of an arbitrary set of elements. □

Exercise 4.3.10. Following the pattern of Definition 4.3.5 (and of Example 4.1.46), define the institution $\mathbf{INS}^{data}$ by adding data constraints as additional sentences to **INS**. □

Note that nowhere in the above has it been assumed that initial models of presentations actually exist in general (nor that the reduct functor used in Definition 4.3.8

has a left adjoint). We do know that in some institutions (for example, in the institution **EQ** of equational logic and in the institution **PEQ** of partial equational logic) any set of sentences over a given signature has an initial model (see Theorem 2.5.14 for the case of **EQ**). On the other hand, there are institutions in which some sets of sentences do not have initial models; the institution **FOEQ** of first-order logic with equality is an example (see Example 2.7.11). Nevertheless, the above definitions work for an arbitrary institution. If a set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences has no initial model, then an initiality constraint **initial** Φ **through** θ based on this set has no model, even if the class $Mod_\Sigma(\Phi)$ of models of this set of sentences is not empty.

Exercise 4.3.11. Any set of sentences in the equational institution **EQ** has a model, and moreover, it has an initial model. Show that neither of these properties carries over to the institution $\mathbf{EQ}^{init}$ of initiality constraints in **EQ**. That is, give a presentation in $\mathbf{EQ}^{init}$ that has no model. □

Exercise 4.3.12. Recall the institution **Horn** of Horn formulae from Exercise 4.1.21 and show that every set of sentences in **Horn** has an initial model. Discuss the interpretation of predicates in initial models: notice that they hold "minimally", meaning that only positive cases need to be explicitly specified. Extend this analysis to data constraints, and use this to specify the transitive and reflexive closure of an arbitrary binary predicate. □

Exercise 4.3.13. Working in the institution **EQ** as in Exercise 4.3.7, follow Definition 4.3.8 and define *generation constraints* **generated over** σ **through** θ that are satisfied by algebras A such that $A|_\theta$ is generated in a suitable sense by $A|_{\sigma;\theta}$. Define an institution $\mathbf{EQ}^{gen}$ extending **EQ** by generation constraints.

Assuming that each category of models in **INS** comes equipped with a factorisation system (Section 3.3), introduce generation constraints for **INS** anticipating Definition 4.5.1 and extend **INS** correspondingly. □

Exercise 4.3.14. Following Exercise 3.5.24, dualise the concept of data constraint. A *co-data constraint* in an institution **INS** can be written as **co-data** Φ' **over** σ **through** θ, where Φ', σ and θ are as in Definition 4.3.8. A Σ-model $M \in |\mathbf{Mod}(\Sigma)|$ *satisfies* **co-data** Φ' **over** σ **through** θ if $M|_\theta$ is a cofree model of Φ' w.r.t. the reduct functor $\_|_\sigma : \mathbf{Mod}[\langle \Sigma', \Phi' \rangle] \to \mathbf{Mod}(\Sigma_1)$ over its σ-reduct, with the identity as counit, that is, if $M|_\theta \models_{\Sigma'} \Phi'$ and for any $M' \in Mod_{\Sigma'}(\Phi')$ and Σ_1-morphism $f : M'|_\sigma \to M|_{\sigma;\theta}$ there exists a unique Σ'-morphism $f^\# : M' \to M|_\theta$ such that $f^\#|_\sigma = f$. Extend this definition to build an institution $\mathbf{INS}^{codata}$ by adding co-data constraints as additional sentences to **INS**.

Discuss the use of co-data constraints in standard institutions like **EQ** and **FOPEQ**. For instance, consider the following simple presentation:

$$
\begin{array}{l}
\textbf{spec}\ \textsc{Stream} = \textbf{sorts}\ \mathit{elem}, \mathit{stream} \\
\qquad\qquad\quad \textbf{ops}\ \ \mathit{hd}\colon \mathit{stream} \to \mathit{elem} \\
\qquad\qquad\qquad\quad \mathit{tl}\colon \mathit{stream} \to \mathit{stream} \\
\qquad\qquad\qquad\quad \mathit{cons}\colon \mathit{elem} \times \mathit{stream} \to \mathit{stream} \\
\qquad\qquad\quad \forall x\colon \mathit{elem}, \forall s\colon \mathit{stream} \\
\qquad\qquad\qquad\quad \bullet\ \mathit{hd}(\mathit{cons}(x,s)) = x \\
\qquad\qquad\qquad\quad \bullet\ \mathit{tl}(\mathit{cons}(x,s)) = s
\end{array}
$$

Check that any model M of $\textsc{Stream}$ that is cofree over $E = |M|_{\mathit{elem}}$ (w.r.t. the reduct functor given by the obvious signature inclusion) is isomorphic to the algebra E^ω of (countably) infinite streams of elements from E, with the operations defined in the standard way.

Much the same effect is achieved even when we remove the operation *cons* and the two axioms from this presentation: check that if Σ is a signature with sorts $\mathit{elem}, \mathit{stream}$ and operations $\mathit{hd}\colon \mathit{stream} \to \mathit{elem}$, $\mathit{tl}\colon \mathit{stream} \to \mathit{stream}$ then cofree Σ-models over their carrier E of sort *elem* are (up to isomorphism) the algebras E^ω of (countably) infinite streams of elements from E, with *hd* and *tl* defined in the standard way. Check then that in any such algebra the two axioms in $\textsc{Stream}$ define the operation *cons* unambiguously. □

4.4 Exact institutions

As illustrated in Sections 4.2 and 4.3, institutions provide a sufficient basis for much of the standard machinery of specifications without the need for further assumptions. Still, the structure and properties of a logical system exposed by the definition of an institution are very limited, and do not provide an adequate basis for many other aspects of the theory and practice of software specification and development. As discussed in the introduction to this chapter, this should not discourage us from working within the institutional framework. On the contrary, it is worth trying to find some adequately abstract additional assumptions that are sufficient for the purpose at hand. As always in mathematics, the main informal guideline to follow is to keep the additional assumptions to a minimum. Part of the payoff is that this forces us to work at a level of generality and abstraction that ensures a deeper understanding of the essence of the studied phenomena, while at the same time covering as many of the cases of potential interest as possible.

In this section and the next we will illustrate this strategy by presenting some extensions to the notion of an institution by additional structure or properties that are required to support study of more detailed properties of specifications.

The ways in which specifications (or programs, systems, or structures of any kind) are put together is the very essence of the theory and methodology of software specification and development. One of the basic tools for "putting things together" is the categorical notion of colimit (cf. Section 3.2) with pushouts as a particularly important special case; see for instance Section 6.3 below. Putting specifications

together then involves taking colimits in the category of theories. It would be rather inconvenient to have to establish the existence of a colimit for each diagram of interest separately, so we normally require the category of theories to be cocomplete (or at least finitely cocomplete). Checking this directly would be tedious — and this is why the following general result is useful.

Theorem 4.4.1. *For any institution* **INS**, *if the category* $\mathbf{Sign}_{\mathbf{INS}}$ *of signatures in* **INS** *is cocomplete then so is the category* $\mathbf{Th}_{\mathbf{INS}}$ *of theories in* **INS**.

Proof. Let D be a diagram in $\mathbf{Th}_{\mathbf{INS}}$ with $|G(D)|_{node} = N$ and $D_n = \langle \Sigma_n, \Phi_n \rangle$ for $n \in N$. Let D' be the corresponding diagram in $\mathbf{Sign}_{\mathbf{INS}}$, hence $D'_n = \Sigma_n$ for $n \in N$. By the assumption of the theorem, D' has a colimit, say $\langle \alpha_n : \Sigma_n \to \Sigma \rangle_{n \in N}$. Let $\Phi = Cl_\Sigma(\bigcup_{n \in N} \alpha_n(\Phi_n))$. Then for each $n \in N$, $\alpha_n : \langle \Sigma_n, \Phi_n \rangle \to \langle \Sigma, \Phi \rangle$ is a theory morphism (this is obvious) and $\langle \alpha_n \rangle_{n \in N}$ is a colimit of D in $\mathbf{Th}_{\mathbf{INS}}$. For: first notice that it is a cocone on D (since it is a cocone on D' in $\mathbf{Sign}_{\mathbf{INS}}$), and then consider another cocone on D, say $\langle \beta_n : \langle \Sigma_n, \Phi_n \rangle \to \langle \Sigma', \Phi' \rangle \rangle_{n \in N}$. By the construction, there exists a unique signature morphism $\sigma : \Sigma \to \Sigma'$ such that for each $n \in N$, $\alpha_n ; \sigma = \beta_n$. To complete the proof, it is sufficient to show that $\sigma : \langle \Sigma, \Phi \rangle \to \langle \Sigma', \Phi' \rangle$ is a theory morphism. By Proposition 4.2.24, it is enough to show that $\sigma(\bigcup_{n \in N} \alpha_n(\Phi_n)) \subseteq \Phi'$. This easily follows from the fact that for each $n \in N$, β_n is a theory morphism, and hence $\sigma(\alpha_n(\Phi_n)) = (\alpha_n ; \sigma)(\Phi_n) = \beta_n(\Phi_n) \subseteq \Phi'$. □

The above proof shows that in fact a stronger property holds: in any institution, the category of theories has all of the colimits that the category of signatures has: the forgetful functor mapping theories to their underlying signatures *lifts colimits*. So, for instance:

Corollary 4.4.2. *For any institution* **INS**, *if the category* $\mathbf{Sign}_{\mathbf{INS}}$ *of signatures in* **INS** *is finitely cocomplete then so is the category* $\mathbf{Th}_{\mathbf{INS}}$ *of theories in* **INS**. □

Notice that the above theorem applies to *any* institution, regardless of the means used to construct it. Hence, for example, if the category $\mathbf{Sign}_{\mathbf{INS}}$ of signatures in an institution **INS** is cocomplete, then not only is the category $\mathbf{Th}_{\mathbf{INS}}$ of theories in **INS** cocomplete, but so are the categories $\mathbf{Th}_{\mathbf{INS}^{init}}$, $\mathbf{Th}_{\mathbf{INS}^{data}}$ and $\mathbf{Th}_{\mathbf{INS}^{codata}}$ of theories in the corresponding institutions with initiality constraints, data constraints and co-data constraints respectively (cf. Definition 4.3.5, Exercise 4.3.10 and Exercise 4.3.14).

Exercise 4.4.3. Assume that the category of signatures of a certain institution has an initial object. What is then an initial object in the category of theories? □

Example 4.4.4. Working in the institution **EQ** of equational logic, recall Example 3.2.35 of a simple pushout of algebraic signatures, and the set $\Phi\mathrm{NAT}$ of equational axioms over the signature $\Sigma\mathrm{NAT}$ given in Exercise 2.5.4. Let $T\mathrm{NAT}$ be the $\Sigma\mathrm{NAT}$-theory presented by $\Phi\mathrm{NAT}$. Let $T\mathrm{NAT}_{fib}$ be the $\Sigma\mathrm{NAT}_{fib}$-theory presented by the axioms $\Phi\mathrm{NAT}_{fib}$ that include $\Phi\mathrm{NAT}$ plus the following:

$fib(0) = succ(0)$
$fib(succ(0)) = succ(0)$
$\forall n{:}nat \bullet fib(succ(succ(n))) = fib(succ(n)) + fib(n)$

Finally, let $T\mathrm{NAT}_{mult}$ be the $\Sigma\mathrm{NAT}_{mult}$-theory presented by the axioms $\Phi\mathrm{NAT}_{mult}$ that include $\Phi\mathrm{NAT}$ plus the following:

$\forall n{:}nat \bullet mult(0,n) = 0$
$\forall n,m{:}nat \bullet mult(succ(n),m) = mult(n,m) + m$

Now, we have theory inclusions:

$$T\mathrm{NAT}_{fib} \hookleftarrow T\mathrm{NAT} \hookrightarrow T\mathrm{NAT}_{mult}$$

with the corresponding signature inclusions given in Example 3.2.35. Their pushout is the $\Sigma\mathrm{NAT}_{fib,mult}$-theory $T\mathrm{NAT}_{fib,mult}$ presented by the union of $\Phi\mathrm{NAT}$, $\Phi\mathrm{NAT}_{fib}$ and $\Phi\mathrm{NAT}_{mult}$.

As in Example 3.2.35, this is deceptively simple, as only single-sorted theory inclusions that introduce different operation names are involved.

Exercise. Give examples of pushouts in the category of equational theories with signatures involving more than one sort, extensions with overlapping sets of operation names, and theory morphisms that are not injective on sort and/or on operation names. Notice however that the extra complications come only from the construction of signature pushouts; the theories are defined in much the same way.

Exercise. Obviously, when giving the set of axioms for $T\mathrm{NAT}_{fib,mult}$, $\Phi\mathrm{NAT}$ may be omitted, as it is already included in the other sets of axioms. Try to generalise this remark to "optimise" the construction of the colimit in the category of theories given in the proof of Theorem 4.4.1. □

We have seen how the assumption that the category of signatures of an institution is (finitely) cocomplete ensures that the institution provides means for "putting theories together". It is also interesting to investigate how this relates to "putting models together", which is what structured programming in the large is all about. There is an important difference here: in the above, and in general when dealing with specifications, we were interested in combining theories, i.e., sets of sentences. In model-theoretic terms, this corresponds to combining classes of models. However, when the specified system is being built, we are interested in expanding and combining *individual* models.

Example 4.4.5. Recall Example 4.4.4 of a simple pushout in the category of theories of the institution **EQ** of equational logic. Consider an arbitrary model N of $T\mathrm{NAT}$, any $\Sigma\mathrm{NAT}_{mult}$-algebra N_2 built by adding to N an interpretation of *fib* such that the axioms in $\Phi\mathrm{NAT}_{fib}$ are satisfied, and any $\Sigma\mathrm{NAT}_{mult}$-algebra N_2 built by adding to N an interpretation of *mult* such that the axioms in $\Phi\mathrm{NAT}_{mult}$ are satisfied. Then, much as in Example 3.4.35 where specific such algebras were considered, N_1 and N_2 may be uniquely combined to a $\Sigma\mathrm{NAT}_{fib,mult}$-algebra N' that expands them

both. The key property now is that the algebras built in this way are models of the theory $T\mathrm{NAT}_{fib,mult}$, and moreover, that all its models may be built in this way. □

It turns out that the crucial link which ensures that constructions to combine theories and to combine models work together smoothly, as in the above example, is the continuity of the model functor in the underlying institution.

Definition 4.4.6 (Exact institution). An institution **INS** is (*finitely*) *exact* if its category of signatures $\mathbf{Sign}_{\mathbf{INS}}$ is (finitely) cocomplete and its model functor $\mathbf{Mod}_{\mathbf{INS}}: \mathbf{Sign}_{\mathbf{INS}}^{op} \to \mathbf{Cat}$ is (finitely) continuous, mapping (finite) colimits in $\mathbf{Sign}_{\mathbf{INS}}$ to limits in **Cat**. □

Example 4.4.7. All of the institutions defined in the examples and sketched in the exercises in Section 4.1.1, with the major exception of **FPL** (Example 4.1.25) and perhaps those given in Examples 4.1.35, 4.1.36 and 4.1.37 where we know nothing about the signature categories, are exact. See Exercises 3.2.53 and 3.4.33 for the standard algebraic case of the equational institution **EQ** — all of the other cases require a similar argument. □

Exercise 4.4.8. The abstract formulation of exactness above may somewhat hide the role of this property in "putting models together". Consider an exact institution **INS** and a diagram D in $\mathbf{Sign}_{\mathbf{INS}}$ with colimit signature Σ'. Anticipating the crucial case of preservation of signature pushouts treated in Definition 4.4.12, show that (up to isomorphism of categories) $\mathbf{Mod}_{\mathbf{INS}}(\Sigma')$ can be defined as follows, where N is the set of nodes in D:

- Σ'-models are families $\langle M_n \in |\mathbf{Mod}_{\mathbf{INS}}(D_n)| \rangle_{n \in N}$ that are compatible with signature morphisms in D in the sense that $M_n = M_m|_{D_e}$ for each edge $e: n \to m$ in the graph of D; and
- Σ'-morphisms between any such Σ'-models $\langle M_n \rangle_{n \in N}$ and $\langle M'_n \rangle_{n \in N}$ are families $\langle h_n: M_n \to M'_n \rangle_{n \in N}$ of morphisms in $\mathbf{Mod}_{\mathbf{INS}}(D_n)$, $n \in N$, that are compatible with signature morphisms in D in the sense that $h_n = h_m|_{D_e}$ for each edge $e: n \to m$ in the graph of D.

Moreover, for each $n \in N$, the reduct functor w.r.t. the colimit injection from D_n to Σ' is just the projection of such families on the n-th component.

HINT: Use Exercise 3.4.32 (and indirectly Exercise 3.2.53). □

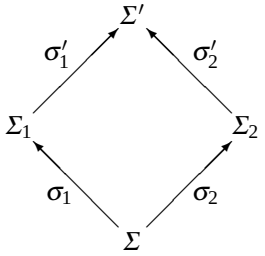
Exercise 4.4.9. Consider a finitely exact institution. Present initiality constraints (Definition 4.3.3) as a special case of data constraints (Definition 4.3.8). Is the assumption that the institution is finitely exact essential? □

Exercise 4.4.10. An interesting standard institution with a cocomplete category of signatures and a model functor that preserves "nearly all" finite colimits of signatures is the institution **SSEQ** of single-sorted equational logic. Give a precise definition of this institution and indicate which colimits of signature diagrams are not preserved by the model functor. HINT: Consider the initial single-sorted signature. □

Definition 4.4.11 (Semi-exact institution). An institution **INS** is *semi-exact* if all pushouts exist in its category of signatures $\mathbf{Sign}_{\mathbf{INS}}$ and its model functor $\mathbf{Mod}_{\mathbf{INS}}\colon \mathbf{Sign}_{\mathbf{INS}}^{op} \to \mathbf{Cat}$ preserves pushouts, mapping them to pullbacks in **Cat**. □

A consequence of the assumption that the model functor of an institution preserves signature pushouts is the well-known *Amalgamation Lemma*.

Definition 4.4.12 (Amalgamation property). Let $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ be an institution and consider the following diagram in **Sign**:



This diagram *admits amalgamation* if:

- for any two models $M_1 \in |\mathbf{Mod}(\Sigma_1)|$ and $M_2 \in |\mathbf{Mod}(\Sigma_2)|$ such that $M_1|_{\sigma_1} = M_2|_{\sigma_2}$, there exists a unique model $M' \in |\mathbf{Mod}(\Sigma')|$ such that $M'|_{\sigma'_1} = M_1$ and $M'|_{\sigma'_2} = M_2$ (we call such M' the *amalgamation* of M_1 and M_2); and
- for any two model morphisms $f_1\colon M_{11} \to M_{12}$ in $\mathbf{Mod}(\Sigma_1)$ and $f_2\colon M_{21} \to M_{22}$ in $\mathbf{Mod}(\Sigma_2)$ such that $f_1|_{\sigma_1} = f_2|_{\sigma_2}$, there exists a unique model morphism $f'\colon M'_1 \to M'_2$ in $\mathbf{Mod}(\Sigma')$ such that $f'|_{\sigma'_1} = f_1$ and $f'|_{\sigma'_2} = f_2$ (we call such f' the *amalgamation* of f_1 and f_2).

The institution **INS** *has the amalgamation property* if all pushouts in **Sign** exist and every pushout diagram in **Sign** admits amalgamation. □

Exercise 4.4.13. Show that if a diagram as in Definition 4.4.12 admits amalgamation and is commutative then all models and morphisms in $\mathbf{Mod}(\Sigma')$ are amalgamations of pairs of (compatible) models and morphisms from $\mathbf{Mod}(\Sigma_1)$ and $\mathbf{Mod}(\Sigma_2)$, respectively. □

Lemma 4.4.14 (Amalgamation Lemma). *Any semi-exact institution has the amalgamation property.* □

The proof of the Amalgamation Lemma is based on the construction of pullbacks in **Cat**, cf. Exercise 3.4.32; see also Exercise 3.4.34, which is the same result in the standard algebraic framework. Note that the opposite implication also holds, so semi-exactness is equivalent to the amalgamation property.

Clearly, every exact institution is finitely exact, and every finitely exact institution is semi-exact. However, the last property is strictly weaker: for example, the institution **SSEQ** of single-sorted equational logic is semi-exact, but not finitely exact (see Exercise 4.4.10). In semi-exact institutions coproducts of signatures need

not exist, or if they exist, need not be preserved by the model functor. However, if signature coproducts exist, the colimits for a large interesting class of signature diagrams (exist and) are preserved:

Proposition 4.4.15. *In any semi-exact institution, if the category of signatures has an initial object then it is finitely cocomplete and the model functor maps colimits of all finite non-empty connected diagrams of signatures to limits in* **Cat**.

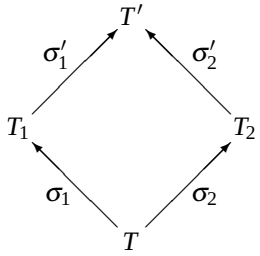
Proof sketch. The first part (existence of colimits of finite signature diagrams) follows as usual, by dualising Exercise 3.2.48; the second part (preservation of limits of finite non-empty connected signature diagrams) follows by Exercise 3.4.55. □

Exercise 4.4.16. Define institutions: **SSFOPEQ** of single-sorted first-order predicate logic with equality, **SSPFOPEQ** of single-sorted partial first-order predicate logic with equality, **SSCEQ** of single-sorted equational logic for continuous algebras, etc. Check that all of these institutions have cocomplete categories of signatures and are semi-exact. However, check that their model functors do not map coproducts of their signatures to products of the corresponding model categories, so these institutions are not (finitely) exact. □

Exercise 4.4.17. Let **INS** be a (finitely) exact institution. Recall that there is a functor $\mathbf{Mod_{Th}}:\mathbf{Th}_{\mathbf{INS}}^{op} \to \mathbf{Cat}$ mapping theories to their model categories and theory morphisms to the corresponding reduct functors (cf. Exercise 4.2.26). Prove that $\mathbf{Mod_{Th}}$ preserves (finite) limits.

HINT: First use the satisfaction condition for **INS** and the Amalgamation Lemma for signatures (Lemma 4.4.14) to prove the following generalisation of the Amalgamation Lemma:

Lemma (Amalgamation Lemma for theories). *Let* **INS** *be a semi-exact institution. Consider a pushout in the category* $\mathbf{Th}_{\mathbf{INS}}$ *of theories:*

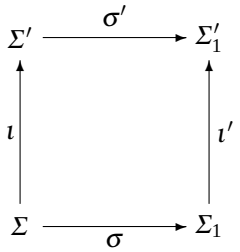


Then, for any two models $M_1 \in Mod[T_1]$ *and* $M_2 \in Mod[T_2]$ *such that* $M_1|_{\sigma_1} = M_2|_{\sigma_2}$, *there exists a unique model* $M' \in Mod[T']$ *such that* $M'|_{\sigma'_1} = M_1$ *and* $M'|_{\sigma'_2} = M_2$, *and similarly for morphisms.*

To complete the proof that $\mathbf{Mod_{Th}}$ is finitely continuous, by Exercise 3.2.48 it is enough to consider the initial theory and its category of models. To show that it is continuous, by Exercise 3.4.23 it is enough to consider coproducts of arbitrary families of theories and their categories of models. □

The trouble with **FPL** and with other institutions based on derived signature morphisms (see Exercise 4.1.23) is more severe than with single-sorted institutions: they are not semi-exact since not all pushouts exist in their signature categories, see Exercise 3.2.54. This motivates the following relaxation of semi-exactness, which is important for applications later on.

Definition 4.4.18 (I-semi-exact institution). For any institution **INS**, we say that a collection **I** of signature morphisms in **INS** is *closed under pushouts* if **I** contains all the identities, is closed under composition (so that **I** is a wide subcategory of $\mathbf{Sign}_{\mathbf{INS}}$) and for any signature morphism $\sigma: \Sigma \to \Sigma_1$ and "**I**-extension of Σ" $\iota: \Sigma \to \Sigma'$ in **I**, there is a pushout in **Sign**



such that $\iota' \in \mathbf{I}$.

Moreover, if all such pushouts with $\iota, \iota' \in \mathbf{I}$ admit amalgamation (i.e., the model functor maps them to pullbacks in **Cat**) we say that **INS** is *semi-exact w.r.t.* **I** (or **I**-*semi-exact*). □

Exercise 4.4.19. As mentioned above, institutions with derived signature morphisms do not have cocomplete signature categories. Check, however, that for example the institution $\mathbf{GEQ}^{der}$ is semi-exact w.r.t. the class of all inclusions (where inclusions are derived signature morphisms that map any n-ary operation name f to the term $f(\boxed{1}, \ldots, \boxed{n})$, cf. Definition 1.5.14). Similarly, check that $\mathbf{GEQ}^{der}$ is semi-exact w.r.t. the class of inclusions that introduce only new constants. (Notice that in general an institution may be **I**-semi-exact without being $\mathbf{I}'$-semi-exact for some $\mathbf{I}' \subseteq \mathbf{I}$.)

For **FPL**, consider the class $\mathbf{I}_{\mathbf{FPL}}$ of signature morphisms $\delta: \mathsf{SIG} \to \mathsf{SIG}'$ that are injective renamings of sort and operation names such that no new value constructors are added for "old" sorts (i.e. sorts in $\delta(\mathsf{SIG})$). Show that **FPL** is $\mathbf{I}_{\mathbf{FPL}}$-semi-exact. Notice that both parts of the assumption on these morphisms are essential. Give an example of a non-injective renaming that does not have a pushout with another **FPL** signature morphism. Give an example of an injective renaming that adds value constructors for an old sort and does not have a pushout with another **FPL** signature morphism. Finally, give an example of a pushout in the the category of **FPL**-signatures that is not mapped by the **FPL**-model functor to a pullback in **Cat**. HINT: Consider two morphisms that add a new sort and a new unary value constructor for a previously unconstrained sort, with the new sort as its argument sort. □

Exercise 4.4.20. To complete the formal picture, note that the category of theories in **FPL** is cocomplete even though its category of signatures is not. Discuss why this is not useful for combining models over different signatures. HINT: Consider a

simple signature with one sort and one binary operation, and two morphisms which map this operation to the projections on the first and second argument respectively. Then these two morphisms do not have a coequaliser in $\mathbf{Sign}_{\mathbf{FPL}}$ while in $\mathbf{Th}_{\mathbf{FPL}}$ their coequaliser is obtained by adding an equation to assert that the two projections coincide. □

We have introduced and studied amalgamation, exactness and semi-exactness as purely technical properties of institutions. However, as hinted at by Example 4.4.5 and the examples it builds on, amalgamation, and hence semi-exactness and exactness, provide a fundamental tool for combining models over different signatures. The point is easiest to see in institutions with standard signatures, like **FOPEQ** or **EQ**, when all the morphisms are inclusions. In that case, generalising the simple example of natural numbers and their extensions by the Fibonacci function and multiplication in Example 3.2.35, given signatures Σ_1 and Σ_2 with $\Sigma = \Sigma_1 \cap \Sigma_2$, we get $\Sigma' = \Sigma_1 \cup \Sigma_2$ as the pushout signature. Now, the amalgamation property ensures that, given a Σ_1-model M_1 and a Σ_2-model M_2 which give the same interpretation to all of the common symbols (in Σ), we can put them together in the obvious way (generalising Example 4.4.5) to interpret all of the symbols in the combined signature Σ'. In the institutional context, this intuition applies as well, but the sharing requirement is expressed by insisting on a common reduct along the indicated signature morphisms, and the combined signature is obtained using the pushout.

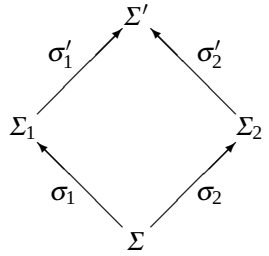
4.4.1 Abstract model theory

One of the ideas behind the definition of institution is that it is important to indicate over which signature one is working. In classical logic, there are a number of theorems in which the signature (or *language*, as logicians would say) over which formulae are constructed must be considered. Here is an example (for this, and for a classical formulation of the Robinson consistency theorem mentioned below, see e.g. [CK90]):

Theorem (Craig interpolation theorem). *In first-order logic, for any two formulae φ_1 and φ_2, if $\varphi_1 \models \varphi_2$ then there exists a formula θ using only the common symbols of φ_1 and φ_2 — that is, those symbols that occur in both formulae — such that $\varphi_1 \models \theta$ and $\theta \models \varphi_2$.* □

In our view, this standard formulation is not very elegant: referring to "the common symbols of φ_1 and φ_2" feels rather clumsy, even though it is easy enough to make it precise in the case of first-order logic. In the institutional framework this can be expressed in a more general and abstract way using colimits in the category of signatures.

Definition 4.4.21 (Craig interpolation property). Let **INS** be an institution with a finitely cocomplete category **Sign** of signatures. **INS** satisfies the *Craig interpolation property* if for any pushout

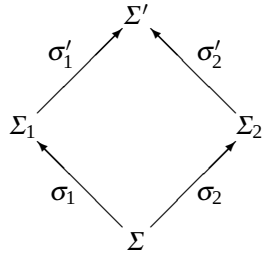


in **Sign**, and for any Σ_1-sentence $\varphi_1 \in \mathbf{Sen}(\Sigma_1)$ and Σ_2-sentence $\varphi_2 \in \mathbf{Sen}(\Sigma_2)$, if $\sigma_1'(\varphi_1) \models_{\Sigma'} \sigma_2'(\varphi_2)$ then there exists a Σ-sentence $\theta \in \mathbf{Sen}(\Sigma)$ (called an *interpolant* for φ_1 and φ_2) such that $\varphi_1 \models_{\Sigma_1} \sigma_1(\theta)$ and $\sigma_2(\theta) \models_{\Sigma_2} \varphi_2$. □

Not only has "the common symbols of φ_1 and φ_2" been captured by the simple categorical concept of a pushout here, but we were also forced to identify the signatures over which the individual consequence relations are considered. In our view, this is a much improved statement of the Craig interpolation property! Not only does it seem more clear (of course, any comparison should be made with a fully formal statement of the Craig interpolation theorem in the classical framework, not with the presentation given above), it is also more abstract and may be used for any logical system formalised as an institution, not just for first-order logic.

Here is another example, which states that consistent extensions of a complete theory (cf. Definition 4.2.27) combine safely:

Definition 4.4.22 (Robinson consistency property). Let **INS** be an institution with a finitely cocomplete category **Sign** of signatures. **INS** satisfies the *Robinson consistency property* if for any pushout



in **Sign**, and for any complete Σ-theory $T = \langle \Sigma, \Phi \rangle$ and consistent theories $T_1 = \langle \Sigma_1, \Phi_1 \rangle$ and $T_2 = \langle \Sigma_2, \Phi_2 \rangle$ such that $\sigma_1 : T \to T_1$ and $\sigma_2 : T \to T_2$ are theory morphisms, the Σ'-presentation $\langle \Sigma', \sigma_1'(\Phi_1) \cup \sigma_2'(\Phi_2) \rangle$ is consistent. □

Exercise 4.4.23. Adapt any standard proof of the Craig interpolation theorem to show that **FOPEQ** has the Craig interpolation property for those pushouts where at least one of σ_1 or σ_2 is injective on sorts. Construct a counterexample which shows that the proof must break down if neither σ_1 nor σ_2 is injective on sort names (injectivity on operation and predicate names does not have to be required). HINT: See [Bor05].

Show also that the Craig interpolation theorem for **FOPEQ** implies the analogous result for some of the subinstitutions of **FOPEQ** (see Exercise 4.1.13), for

instance for **FOEQ**. Note though that your argument will not work for **FOP**, first-order predicate logic without equality — in fact, Craig interpolation may fail in **FOP** when one of the morphisms involved is non-injective on operation names, even if all the morphisms are injective on sort names. Of course, the standard proofs of Craig interpolation easily adapt to **FOP** when the morphisms involved are injective (on sort names as well as on operation names). □

It is well known that equational logic does not have the interpolation property:

Counterexample 4.4.24. In **EQ**, consider the signature Σ with three sorts s, s_1 and s_2, and two constants $a,b:s$. Let Σ_1 and Σ_2 extend Σ by a constant $e:s_1$ and by a unary operation $f:s_1 \to s_2$ respectively. Let Σ' be the union of Σ_1 and Σ_2 (this is the pushout signature for the two signature inclusions). Consider the sentences $\forall x{:}s_2 \bullet a = b \in \mathbf{Sen_{EQ}}(\Sigma_1)$ and $a = b \in \mathbf{Sen_{EQ}}(\Sigma_2)$. Clearly, over Σ' we have $\forall x{:}s_2 \bullet a = b \models a = b$ (since all Σ'-algebras have non-empty carriers for all sorts).

Suppose that we have an interpolant $\theta \in \mathbf{Sen_{EQ}}(\Sigma)$ for $\forall x{:}s_2 \bullet a = b$ and $a = b$, so that $\forall x{:}s_2 \bullet a = b \models \theta$ over Σ_1 and $\theta \models a = b$ over Σ_2. Consider a Σ_1-algebra A_1 with the carrier of sort s_2 empty and with $a_{A_1} \neq b_{A_1}$. Clearly, $A_1 \models_{\Sigma_1} \forall x{:}s_2 \bullet a = b$, and so also $A_1 \models_{\Sigma_1} \theta$. Hence, $A_1|_\Sigma \models_\Sigma \theta$. Take a subalgebra of $A_1|_\Sigma$ with the empty carrier of sort s_1, which satisfies θ, and consider its expansion A_2 to a Σ_2-algebra. Then $A_2 \models_{\Sigma_2} \theta$ but $A_2 \not\models_{\Sigma_2} a = b$. Contradiction. □

Exercise 4.4.25. It is often stated that equational logic has interpolation (at least for pushouts w.r.t. injective signature morphisms) if one admits a *set of interpolants*, rather than just a single interpolant sentence θ as in Definition 4.4.21. Spell out this property following Definition 4.4.21, but using a set of sentences $\Theta \subseteq \mathbf{Sen}(\Sigma)$ in place of a single sentence $\theta \in \mathbf{Sen}(\Sigma)$. It also makes sense then to replace the single sentence $\varphi_1 \in \mathbf{Sen}(\Sigma_1)$ by a set $\Phi_1 \subseteq \mathbf{Sen}(\Sigma_1)$.

Unfortunately, equational logic has this property only if we restrict attention to algebras with non-empty carriers for all sorts. Carry out the proof for this case assuming that the signature morphisms considered are injective (HINT: see [Rod91]) and note where the assumption that the carriers are non-empty is important. Give a counterexample which shows that in general no single interpolant can be sufficient here. Extend this proof to the case where only one of the signature morphisms is injective on sorts (HINT: see [RG00], [PŞR09]).

Check that Counterexample 4.4.24 shows that the institution **EQ** of equational logic (with models that admit empty carriers) does not have the interpolation property, not even when sets of interpolants are allowed (and the morphisms involved are signature inclusions).

Go through other examples of institutions in Section 4.1.1 and check which of them have the interpolation property, either with a single interpolant, or with a set of interpolants (at least for pushouts involving signature inclusions, where this notion makes sense). □

Of course, we cannot expect to be able to prove that either the Craig interpolation or Robinson consistency properties are satisfied by an arbitrary institution

— they simply do not hold for some logics. However, one may attempt to identify other conditions on the underlying institution which imply the two properties. Along these lines, under some further technical assumptions, the two properties are equivalent: an institution satisfying certain technical assumptions satisfies the Craig interpolation property if and only if it satisfies the Robinson consistency property. This reflects what is well-known in classical model theory, where the two properties are indeed derivable from one another.

4.4.2 Free variables and quantification

In logic, formulae may contain free variables; such formulae are called *open*, as opposed to *closed* formulae which have no free variables. To interpret an open formula, one needs not only an interpretation for the symbols of the underlying signature (a model) but also an interpretation for the free variables (a valuation of variables in the model). This provides a natural way to deal with quantifiers. The need for open formulae also arises in the study of specification languages. In fact, we will use them to abstractly express the basic notion of behavioural equivalence in Section 8.5.3, see Exercise 8.5.61.

Fortunately we do not have to change the notion of an institution to cope with free variables — we can provide open formulae in the present framework. Note that we use here the term "formula" rather than "sentence", which is reserved for the sentences of the underlying institution, corresponding to closed formulae.

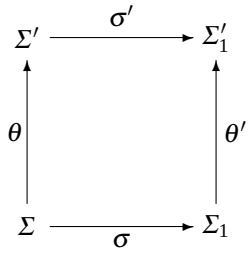
Consider the institution **GEQ** of ground equational logic (Example 4.1.3). Let $\Sigma = \langle S, \Omega \rangle$ be an algebraic signature. For any S-indexed family of sets, $X = \langle X_s \rangle_{s \in S}$, define $\Sigma(X)$ to be the extension of Σ by the elements of X as new constants of the appropriate sorts. Any sentence over $\Sigma(X)$ may be viewed as an open formula over Σ with free variables X. Given a Σ-algebra A, to determine whether an open Σ-formula with variables X holds in A we have to first fix a valuation of variables X into $|A|$. Such a valuation corresponds exactly to an expansion of A to a $\Sigma(X)$-algebra.

Given a translation of sentences along an algebraic signature morphism $\sigma: \Sigma \to \Sigma'$ we can extend it to a translation of open formulae: we translate an open Σ-formula with variables X, which is a $\Sigma(X)$-sentence, to the corresponding $\Sigma'(X')$-sentence, which is an open Σ'-formula with variables X'. Here X' results from X by an appropriate renaming of sorts determined by σ (we also have to avoid unintended "clashes" of variables and operation symbols).

The above ideas generalise to any semi-exact institution $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$.

Definition 4.4.26 (Open formula). Let $\Sigma \in |\mathbf{Sign}|$ be a signature in **INS**. Any pair $\langle \varphi, \theta \rangle$, where $\theta: \Sigma \to \Sigma'$ is a signature morphism and $\varphi \in \mathbf{Sen}(\Sigma')$, is an *open Σ-formula* with variables "$\Sigma' \setminus \theta(\Sigma)$". For any Σ-model $M \in |\mathbf{Mod}(\Sigma)|$, a *valuation* of variables "$\Sigma' \setminus \theta(\Sigma)$" into M is a Σ'-model $M' \in |\mathbf{Mod}(\Sigma')|$ which is a θ-expansion of M, i.e., such that $M'|_\theta = M$. We say that $\langle \varphi, \theta \rangle$ *holds in M under valuation M'* iff

$M' \models_{\Sigma'} \varphi$. If $\sigma: \Sigma \to \Sigma_1$ is a signature morphism then we define the translation of $\langle \varphi, \theta \rangle$ along σ as $\langle \sigma'(\varphi), \theta' \rangle$, where



is a pushout in **Sign**. □

Note the quotation marks around the "set of variables" $\Sigma' \setminus \theta(\Sigma)$ in the above definition: since $\Sigma' \setminus \theta(\Sigma)$ makes no sense in an arbitrary institution, it is only meaningful as an aid to our intuition.

In the standard logical framework there may be no valuation of a set of variables into a model containing an empty carrier. Similarly here, a valuation need not always exist. For example, in **GEQ** if a signature morphism $\theta: \Sigma \to \Sigma'$ is not injective then some Σ-models have no θ-expansion.

There is a rather subtle problem with the above definition: pushouts are defined only up to isomorphism, so strictly speaking the translation of open formulae is not well-defined. The following exercise shows that (at least for semantic analysis) an arbitrary pushout may be selected and so we may safely accept the above definition of translation.

Exercise 4.4.27. Consider an isomorphism $\iota: \Sigma'_1 \to \Sigma''_1$ in **Sign**, with inverse ι^{-1}. Since functors preserve isomorphisms, $\mathbf{Sen}(\iota): \mathbf{Sen}(\Sigma'_1) \to \mathbf{Sen}(\Sigma''_1)$ is a bijection and $\mathbf{Mod}(\iota): \mathbf{Mod}(\Sigma''_1) \to \mathbf{Mod}(\Sigma'_1)$ is an isomorphism in **Cat**. Show that moreover, for any $\psi \in \mathbf{Sen}(\Sigma'_1)$ and $M'_1 \in |\mathbf{Mod}(\Sigma'_1)|$, $M'_1 \models_{\Sigma'_1} \psi \iff M'_1|_{\iota^{-1}} \models_{\Sigma''_1} \iota(\psi)$. □

Sometimes we want to restrict the class of signature morphisms that may be used to construct open formulae. In fact, in the above remarks sketching how free variables may be introduced into **GEQ** we used just algebraic signature inclusions $\iota: \Sigma \hookrightarrow \Sigma'$ where the only new symbols in Σ' were constants. To guarantee that the translation of open formulae is defined under such a restriction, we consider only restrictions to a collection **I** of signature morphisms that is closed under pushouts (see Definition 4.4.18).

Examples of such collections **I** in **AlgSig** include: the collection of all algebraic signature inclusions, the restriction of this to inclusions $\theta: \Sigma \hookrightarrow \Sigma'$ such that Σ' contains no new sorts, the further restriction of this by the requirement that Σ' contains new constants only (as above), the collection of all algebraic signature morphisms which are surjective on sorts, the collection of all identities, and the collection of all morphisms. Note that most of these permit variables denoting operations or even sorts.

4.4.2.1 Universal quantification

In the rest of this section we briefly sketch how to universally close the open formulae introduced above.

Let **I** be a collection of signature morphisms that is closed under pushouts. Let Σ be a signature and let $\langle \varphi, \theta \rangle$ be an open Σ-formula such that $\theta \in \mathbf{I}$. Consider the universal closure of $\langle \varphi, \theta \rangle$, written $\forall \theta \bullet \varphi$, as a new Σ-sentence. The satisfaction relation and the translation of a sentence $\forall \theta \bullet \varphi$ along a signature morphism are defined in the expected way:

- A Σ-model satisfies the Σ-sentence $\forall \theta \bullet \varphi$ if $\langle \varphi, \theta \rangle$ holds in this model under any valuation of the variables "$\Sigma' \setminus \theta(\Sigma)$", that is, for any $M \in |\mathbf{Mod}(\Sigma)|$, $M \models_\Sigma \forall \theta \bullet \varphi$ if for all $M' \in |\mathbf{Mod}(\Sigma')|$ such that $M'|_\theta = M$, $M' \models_{\Sigma'} \varphi$.
- For any signature morphism $\sigma: \Sigma \to \Sigma_1$, $\sigma(\forall \theta \bullet \varphi)$ is $\forall \theta' \bullet \sigma'(\varphi)$, where

$$\begin{array}{ccc} \Sigma' & \xrightarrow{\sigma'} & \Sigma'_1 \\ {\scriptstyle\theta}\uparrow & & \uparrow{\scriptstyle\theta'} \\ \Sigma & \xrightarrow[\sigma]{} & \Sigma_1 \end{array}$$

  is a pushout in **Sign** such that $\theta' \in \mathbf{I}$.

Note that in the above we have extended our underlying institution **INS**. Formally:

Definition 4.4.28 (Institution with universally closed formulae). Let **INS** be an institution, and let **I** be a collection of signature morphisms in **INS** that is closed under pushouts such that **INS** is **I**-semi-exact. The *extension of* **INS** *by universal closure w.r.t.* **I** is the following institution $\mathbf{INS}^{\forall(\mathbf{I})}$:

- $\mathbf{Sign}_{\mathbf{INS}^{\forall(\mathbf{I})}}$ is $\mathbf{Sign}_{\mathbf{INS}}$.
- For any signature Σ, $\mathbf{Sen}_{\mathbf{INS}^{\forall(\mathbf{I})}}(\Sigma)$ is the disjoint union of $\mathbf{Sen}_{\mathbf{INS}}(\Sigma)$ with the collection[20] of all universal closures $\forall \theta \bullet \varphi$ of open Σ-formulae, where $\theta \in \mathbf{I}$; for any signature morphism $\sigma: \Sigma \to \Sigma_1$, $\mathbf{Sen}_{\mathbf{INS}^{\forall(\mathbf{I})}}(\sigma)$ is the function induced by $\mathbf{Sen}_{\mathbf{INS}}(\sigma)$ on $\mathbf{Sen}_{\mathbf{INS}}(\Sigma)$ and by the notion of translation defined above on universally closed open Σ-formulae.
- $\mathbf{Mod}_{\mathbf{INS}^{\forall(\mathbf{I})}}$ is $\mathbf{Mod}_{\mathbf{INS}}$.
- The satisfaction relation in $\mathbf{INS}^{\forall(\mathbf{I})}$ is induced by the satisfaction relation of **INS** for **INS**-sentences and the notion of satisfaction for universally closed open formulae as defined above. □

The following theorem guarantees that $\mathbf{INS}^{\forall(\mathbf{I})}$ is in fact an institution, modulo the above remark about the definition of the translation of open formulae.

[20] As usual, we disregard here the foundational problems which may arise if **I** is not a set.

Theorem 4.4.29 (Satisfaction condition for INS$^{\forall(\mathbf{I})}$). *Let* **INS** *and* **I** *be as in Definition 4.4.28. For any signature morphism $\sigma:\Sigma \to \Sigma_1$, open Σ-formula $\langle \varphi, \theta \rangle$ (where $\theta \in \mathbf{I}$), Σ_1-model $M_1 \in |\mathbf{Mod}(\Sigma_1)|$, and pushout*

$$\begin{array}{ccc} \Sigma' & \xrightarrow{\sigma'} & \Sigma'_1 \\ {\scriptstyle\theta}\uparrow & & \uparrow{\scriptstyle\theta'} \\ \Sigma & \xrightarrow[\sigma]{} & \Sigma_1 \end{array}$$

in **Sign** *such that $\theta' \in \mathbf{I}$,*

$$M_1|_\sigma \models_\Sigma \forall \theta \bullet \varphi \quad \textit{iff} \quad M_1 \models_{\Sigma_1} \forall \theta' \bullet \sigma'(\varphi)$$

Proof.

($\Rightarrow$): Assume that $M_1|_\sigma \models_\Sigma \forall \theta \bullet \varphi$ and let M'_1 be a θ'-expansion of M_1. Put $M' = M'_1|_{\sigma'}$. Obviously, $M'|_\theta = M'_1|_{\theta;\sigma'} = M'_1|_{\sigma;\theta'} = M_1|_\sigma$. Thus, since $M_1|_\sigma \models_\Sigma \forall \theta \bullet \varphi$, $M' \models_{\Sigma'} \varphi$. Hence, by the satisfaction condition of **INS**, $M'_1 \models_{\Sigma'_1} \sigma'(\varphi)$, which proves $M_1 \models_{\Sigma_1} \forall \theta' \bullet \sigma'(\varphi)$.

($\Leftarrow$): Assume that $M_1 \models_{\Sigma_1} \forall \theta' \bullet \sigma'(\varphi)$ and let M' be a θ-expansion of $M_1|_\sigma$. Since **INS** is **I**-semi-exact, there exists a θ'-expansion M'_1 of M_1 such that $M'_1|_{\sigma'} = M'$. Then, since $M_1 \models_{\Sigma_1} \forall \theta' \bullet \sigma'(\varphi)$, $M'_1 \models_{\Sigma'_1} \sigma'(\varphi)$. Thus, by the satisfaction condition, $M' \models_{\Sigma'} \varphi$, which proves $M_1|_\sigma \models_\Sigma \forall \theta \bullet \varphi$. □

Example 4.4.30. Let **I** be the collection of algebraic signature inclusions $\iota:\Sigma \hookrightarrow \Sigma'$ in **AlgSig** such that $\Sigma' \setminus \Sigma$ contains new constants only. The institution **GEQ**$^{\forall(\mathbf{I})}$ essentially coincides with the institution **EQ** of equational logic (modulo the details of the notation used for sentences), as suggested already in Exercise 2.1.6. If $\Sigma' \setminus \Sigma$ is allowed to contain new operation names (not just constants), then quantification along morphisms in **I** leads to a version of second-order logic. □

Other quantifiers (there exists, there exists a unique, there exist infinitely many, for almost all, ...) may be introduced in the same manner as we have just introduced universal quantifiers. Example 4.1.41 illustrates how one may introduce logical connectives. By iterating these constructions one can, for example, derive the institution of first-order logic from the institution of ground atomic formulae.

4.5 Institutions with reachability structure

An alternative to the standard initial algebra approach to specifications is to take the reachable semantics of presentations, as discussed in Section 2.7.2, where from

among all the algebras satisfying a presentation only the *reachable* algebras are selected. In Section 4.3 we argued that it is important to consider not just initial algebras, but more generally, algebras that are free extensions of a specified part; similarly, it is important here to consider not just reachable algebras, but more generally, algebras that are generated by some specified part. Given an algebraic signature Σ and a subsignature $\Sigma' \subseteq \Sigma$, a Σ-algebra A is *reachable from* Σ' if it has no proper subalgebra with the same Σ'-reduct. (**Exercise:** Show that this is the same as to require that the algebra is generated by the set of all its elements in the carriers of the sorts in Σ', as defined in Exercise 1.2.6.) To generalise this notion to the framework of an arbitrary institution we will proceed along the lines suggested by the "categorical theory of reachability" presented in Section 3.3 based on factorisation systems.

Definition 4.5.1 (Reachable model). Let $\langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ be an institution. Assume that for each signature $\Sigma \in |\mathbf{Sign}|$, we have a factorisation system $\langle \mathbf{E}_\Sigma, \mathbf{M}_\Sigma \rangle$ for the category $\mathbf{Mod}(\Sigma)$ of Σ-models.

Let $\sigma: \Sigma' \to \Sigma$ be a signature morphism. A Σ-model $M \in |\mathbf{Mod}(\Sigma)|$ is σ-*reachable* if M has no proper submodel with an isomorphic σ-reduct, that is, if any factorisation monomorphism $m: N \to M$ in $\mathbf{M}_\Sigma$ such that $m|_\sigma$ is an isomorphism in $\mathbf{Mod}(\Sigma')$ is in fact an isomorphism in $\mathbf{Mod}(\Sigma)$. □

Example 4.5.2. Recall that for any algebraic signature $\Sigma \in \mathbf{AlgSig}$, the categories $\mathbf{Alg}(\Sigma)$, $\mathbf{PAlg}(\Sigma)$ and $\mathbf{CAlg}(\Sigma)$ of total, partial and continuous algebras come equipped with factorisation systems (Examples 3.3.3, 3.3.13 and 3.3.14, respectively). Hence, the above definition makes sense in the institutions **EQ** of equational logic, **PEQ** of partial equational logic and **CEQ** of equational logic for continuous algebras, yielding the expected notions. □

Exercise 4.5.3. Recall that by Definition 3.3.7 a Σ-model is reachable if it has no proper submodel. Show that if **INS** is finitely exact then reachability is a special case of σ-reachability as defined above. (HINT: Use the fact that there is an initial signature with the singleton category **1** of models.) □

In Section 3.3 it was shown how the notion of reachability introduced there may be related to an equivalent definition stated in terms of quotients of initial models (Theorem 3.3.8(1)). In the standard algebraic case, an algebra is reachable if and only if it is isomorphic to a quotient of the algebra of ground terms (Exercise 1.4.14). To give an analogous result for σ-reachability we have to be able to build terms over a specified reduct of the given algebra (cf. Exercise 3.5.11). Given such a construction, a Σ-algebra A is reachable from $\Sigma' \subseteq \Sigma$ if and only if evaluation in A of Σ-terms over the Σ'-reduct of A is surjective, or equivalently, if A is a natural quotient of the algebra of Σ-terms built over $A|_{\Sigma'}$. We introduce a generalisation of the construction of term algebras to an arbitrary institution by requiring that reduct functors induced by signature morphisms have left adjoints. Notice that only signatures are involved in this definition, no sentences, and so this requirement indeed corresponds to the mild assumption that free models (term algebras) may be built along arbitrary signature morphisms.

Definition 4.5.4 (Institution with reachability structure). An *institution with reachability structure* is an institution $\langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ together with:

- for each signature $\Sigma \in |\mathbf{Sign}|$, a factorisation system $\langle \mathbf{E}_\Sigma, \mathbf{M}_\Sigma \rangle$ for the category $\mathbf{Mod}(\Sigma)$ of Σ-models; and
- for each signature morphism $\sigma: \Sigma' \to \Sigma$, a *$\sigma$-free functor* $\mathbf{F}_\sigma: \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ which is left adjoint to the σ-reduct functor $\_|_\sigma: \mathbf{Mod}(\Sigma) \to \mathbf{Mod}(\Sigma')$ with unit $\eta^\sigma: \mathbf{Id}_{\mathbf{Mod}(\Sigma')} \to \mathbf{F}_\sigma(\_)|_\sigma$.

(As usual, sub- and superscripts will be omitted when convenient.) □

Example 4.5.5. The institution **EQ** of equational logic equipped with factorisation systems for categories of algebras (cf. Example 3.3.3) has reachability structure — the free functors are given by Exercise 3.5.11. □

Exercise 4.5.6. Show that the institution **PEQ** of partial equational logic with the factorisation systems given by Example 3.3.13 for categories of partial algebras forms an institution with reachability structure. (HINT: Free functors are rather trivial here.)

Similarly, show that the institution **CEQ** of equational logic for continuous algebras with the factorisation systems given by Example 3.3.14 for categories of continuous algebras forms an institution with reachability structure. (HINT: The construction of free functors is much more difficult here — follow the construction for ordinary algebras in Exercise 3.5.11, but when defining the new operations in a free way remember that you have to extend the complete partial order to cover the new values as well, ensuring continuity of the operations.) □

Exercise 4.5.7. Let **INS** be a finitely exact institution. Prove that if every reduct functor in **INS** has a left adjoint, then for every signature Σ the category $\mathbf{Mod}_{\mathbf{INS}}(\Sigma)$ of Σ-models has an initial object. (HINT: Use the fact that there is an initial signature with the singleton category **1** of models.) □

The following theorem generalises well-known facts from the standard algebraic setting. Just like its "predecessor" Theorem 3.3.8, it confirms our confidence in the abstract definitions by showing how their different versions "click together" nicely.

Theorem 4.5.8. *Let* $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ *be an institution with reachability structure. Consider a signature morphism* $\sigma: \Sigma' \to \Sigma$.

1. *A Σ-model $M \in |\mathbf{Mod}(\Sigma)|$ is σ-reachable if and only if it is a natural quotient of the free object over its σ-reduct, that is, the counit morphism $\varepsilon_M = (id_{M|_\sigma})^\# : \mathbf{F}_\sigma(M|_\sigma) \to M$ belongs to $\mathbf{E}_\Sigma$ (cf. Exercise 3.5.24).*
2. *For any σ-reachable model $M \in |\mathbf{Mod}(\Sigma)|$, any model $N \in |\mathbf{Mod}(\Sigma)|$ and Σ'-model morphism $f': M|_\sigma \to N|_\sigma$, there exists at most one Σ-model morphism $f: M \to N$ that extends f' (i.e., such that $f|_\sigma = f'$).*
3. *Every Σ-model has a unique (up to isomorphism) σ-reachable submodel with an isomorphic σ-reduct.*

4. *If $M \in |\mathbf{Mod}(\Sigma)|$ is σ-reachable then for any Σ-model morphism $f: N \to M$ such that $f|_\sigma$ is an isomorphism, f is a factorisation epimorphism (i.e., $f \in \mathbf{E}_\Sigma$).*

Proof.

1. $(\Rightarrow)$: Let $\mathbf{F}_\sigma(M|_\sigma) \xrightarrow{e} N \xrightarrow{m} M$ be a factorisation of $\varepsilon_M: \mathbf{F}_\sigma(M|_\sigma) \to M$. Arguing dually to Exercise 3.5.18 we can show that $m|_\sigma: N|_\sigma \to M|_\sigma$ is an isomorphism. Hence, by the σ-reachability of M, m is an isomorphism, which proves that $\varepsilon_M \in \mathbf{E}_\Sigma$.
   $(\Leftarrow)$: Let $m: N \to M$, $m \in \mathbf{M}_\Sigma$, with $m|_\sigma$ being an isomorphism. Define $f: \mathbf{F}_\sigma(M|_\sigma) \to N$ by $f = ((m|_\sigma)^{-1})^\#$. Then $\eta_{M|_\sigma}; (f; m)|_\sigma = id_{M|_\sigma}$. By the freeness of $\mathbf{F}_\sigma(M|_\sigma)$, this implies that $f; m = \varepsilon_M$. Thus, by the assumption that $\varepsilon_M \in \mathbf{E}_\Sigma$ and by Exercise 3.3.5, m is an isomorphism.
2. Suppose that $f_1, f_2: M \to N$ are such that $f_1|_\sigma = f_2|_\sigma = f'$. Then $\eta_{M|_\sigma}; (\varepsilon_M; f_1)|_\sigma = f' = \eta_{M|_\sigma}; (\varepsilon_M; f_2)|_\sigma$, and so $\varepsilon_M; f_1 = \varepsilon_M; f_2$. Thus, we also have $f_1 = f_2$, since by (1) above ε_M is an epimorphism.
3. Consider an arbitrary Σ-model M. Let $\mathbf{F}_\sigma(M|_\sigma) \xrightarrow{e} N \xrightarrow{m} M$ be a factorisation of $\varepsilon_M: \mathbf{F}_\sigma(M|_\sigma) \to M$. Again, arguing dually to Exercise 3.5.18 we can show that $m|_\sigma: N|_\sigma \to M|_\sigma$ is an isomorphism. Moreover, by the naturality of ε, $\mathbf{F}_\sigma(m|_\sigma); \varepsilon_M = \varepsilon_N; m$, that is $\mathbf{F}_\sigma(m|_\sigma); e; m = \varepsilon_N; m$, and so (since m is a monomorphism) $\varepsilon_N = \mathbf{F}_\sigma(m|_\sigma); e \in \mathbf{E}_\Sigma$. Thus, by (1) again, N is a σ-reachable submodel of M.
   To prove uniqueness up to isomorphism, consider a subobject $m_1: N_1 \to M$ with $m_1|_\sigma$ being an isomorphism and $\varepsilon_{N_1}: F_\sigma(N_1|_\sigma) \to N_1$ in $\mathbf{E}_\Sigma$. Then $\mathbf{F}_\sigma(m_1|_\sigma); \varepsilon_M = \varepsilon_{N_1}; m_1$, and since $\mathbf{F}_\sigma(m_1|_\sigma)$ is an isomorphism, we have two factorisations of $\varepsilon_M: \mathbf{F}_\sigma(M|_\sigma) \to M$, $\langle \mathbf{F}_\sigma(m_1|_\sigma)^{-1}; \varepsilon_{N_1}, m_1 \rangle$ and $\langle e, m \rangle$, which by the uniqueness of factorisations implies that N and N_1 are isomorphic.
4. Let $N \xrightarrow{e} \cdot \xrightarrow{m} M$ be a factorisation of $f: N \to M$. Then, by naturality of ε, $\varepsilon_N; e; m = \mathbf{F}_\sigma(f|_\sigma); \varepsilon_M$. Now, since $f|_\sigma$ (and hence $\mathbf{F}_\sigma(f|_\sigma)$) is an isomorphism, by σ-reachability of M and (1) above, $\varepsilon_N; e; m \in \mathbf{E}_\Sigma$. Thus, by Exercise 3.3.5, m is an isomorphism, and so $f \in \mathbf{E}_\Sigma$. □

4.5.1 The method of diagrams

In the standard algebraic framework, reachable algebras enjoy a number of useful properties which make them especially easy to deal with. As a consequence of the fact that we are able to "name" (using ground terms) all their elements, reachable algebras are easy to describe using the most elementary logical sentences, ground equations. To be more precise: for any algebraic signature Σ and reachable Σ-algebra A, the class

$$Ext(A) = \{B \in |\mathbf{Alg}(\Sigma)| \mid \text{there exists a } \Sigma\text{-homomorphism } h: A \to B\}$$

is definable by the ground Σ-equations that hold in A, that is, $Ext(A) = Mod_{\mathbf{GEQ}}(Th_{\mathbf{GEQ}}(\{A\}))$, and moreover, A is initial in $Ext(A)$. (We will refer to classes of algebras of the form $Ext(A)$ for a reachable algebra A as *ground varieties*.) This gives a one-to-one correspondence between ground equational theories and isomorphism classes of reachable algebras (and furthermore, congruences on ground term algebras by Exercise 1.4.14).

Unfortunately, not all algebras are reachable, and it is clear that this correspondence does not carry over to arbitrary algebras: there are algebras that cannot be characterised as initial models of equational theories. But there is a technical trick that may help: if a Σ-algebra A is not reachable, then consider the signature $\Sigma(A)$ obtained by adding to Σ the elements of $|A|$ as constants of the appropriate sorts. Now, the algebra A has an obvious expansion to a reachable $\Sigma(A)$-algebra $E(A)$, where the new constants are interpreted as the elements they correspond to. This expansion has a number of useful properties:

- Any Σ-homomorphism $h:A \to B$ determines unambiguously an expansion of B to a $\Sigma(A)$-algebra $E_h(B)$ where each new constant in $\Sigma(A)$ is interpreted as the value of h on the corresponding element of $|A|$. Moreover, this expansion is independent from any decomposition of h: for any Σ-homomorphisms $h_1:A \to C$ and $h_2:C \to B$ such that $h = h_1;h_2$, the homomorphism h_2 (or more precisely, its underlying map) is a $\Sigma(A)$-homomorphism from $E_{h_1}(C)$ to $E_h(B)$.
- Intuitively, the expansion does not introduce more structure than necessary to make A reachable; in particular, no new elements are added.

Putting all these together, any Σ-algebra A may be characterised by the set of ground equations on the signature $\Sigma(A)$ that hold in $E(A)$. This technique, known as *the method of diagrams*, is one of the basic tools of classical model theory (cf. e.g. [CK90]). We have already suggested its use in the construction of the free functor corresponding to a signature morphism in Exercise 3.5.11.

In the following the method of diagrams is formulated in the context of an arbitrary institution with reachability structure. We will assume that the institution is finitely exact in order to be able to deal with reachability (not just reachability relative to signature morphisms, cf. Exercises 4.5.3 and 4.5.7).

Definition 4.5.9 (The method of diagrams). Let $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ be a finitely exact institution with reachability structure. **INS** *admits the method of diagrams* if:

- (*Definability of ground varieties*)
  for every signature $\Sigma \in |\mathbf{Sign}|$ and reachable Σ-model $M \in |\mathbf{Mod}(\Sigma)|$, the class

  $$Ext(M) = \{N \in |\mathbf{Mod}(\Sigma)| \mid \text{there exists a } \Sigma\text{-model morphism } h:M \to N\}$$

  of extensions of M is definable, that is, $Ext(M) = Mod_\Sigma(\Phi)$ for some set $\Phi \subseteq \mathbf{Sen}(\Sigma)$.
- (*Existence of diagrams*)
  for every signature $\Sigma \in |\mathbf{Sign}|$ and Σ-model $M \in |\mathbf{Mod}(\Sigma)|$, there exists a signature $\Sigma(M) \in |\mathbf{Sign}|$ and signature morphism $\iota:\Sigma \to \Sigma(M)$ such that:

- M has a reachable ι-expansion $E(M)$: there exists $E(M)$ which is a reachable $\Sigma(M)$-model such that $E(M)\big|_\iota = M$;
- ι-reduct is an isomorphism of the slice categories $\mathbf{Mod}(\Sigma(M))\!\uparrow\! E(M)$ and $\mathbf{Mod}(\Sigma)\!\uparrow\! M$ (see Exercise 3.1.30), that is, for any Σ-model morphism $f\!:\!M \to N$, there exists a unique ι-expansion of N, $E_f(N)$, such that f has an ι-expansion $E(f)\!:\!E(M) \to E_f(N)$ and such that any Σ-model morphism $h\!:\!N \to N_1$ has a unique ι-expansion $E(h)\!:\!E_f(N) \to E_{f;h}(N_1)$; and
- ι-reduct preserves the factorisation system on $\mathbf{Mod}(\Sigma(M))\!\uparrow\! E(M)$ as inherited from $\mathbf{Mod}(\Sigma(M))$, that is, for any $f\!:\!E(M) \to N'$ and $h\!:\!N' \to N''$, if $h \in \mathbf{E}_{\Sigma(M)}$ then $h\big|_\iota \in \mathbf{E}_\Sigma$ and if $h \in \mathbf{M}_{\Sigma(M)}$ then $h\big|_\iota \in \mathbf{M}_\Sigma$.

Then, $\Sigma(M)$ is called the *diagram signature for* M (with *signature inclusion* ι), $E(M)$ is called the *diagram expansion of* M, and finally the theory $\Delta^+(M) = Th_{\Sigma(M)}(Ext(E(M)))$ is called the (*positive*) *diagram of* M. □

Example 4.5.10. The institutions **EQ** of equational logic, **PEQ** of partial equational logic, and **CEQ** of equational logic for continuous algebras admit the method of diagrams. Ground varieties in **EQ** are definable by sets of ground equations; ground varieties of **PEQ** are definable by sets of ground equations and ground definedness formulae; ground varieties in **CEQ** are definable by sets of ground infinitary equations. For any (total, partial, or continuous) Σ-algebra A, the diagram signature for A is formed by adding constants corresponding to all the elements of $|A|$. The diagram expansion of a partial algebra is formed by requiring that the new constants are defined and have the expected values. □

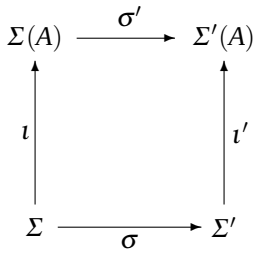
Exercise 4.5.11. Show that in any institution that admits the method of diagrams, and for any model M, the class of models of the positive diagram of M is the class of all extensions of the diagram expansion of M: $Mod_{\Sigma(M)}(\Delta^+(M)) = Ext(E(M))$. □

4.5.2 Abstract algebraic institutions

In Exercise 3.5.11 we suggested the use of the method of diagrams to prove that in the standard algebraic framework, the reduct functor induced by a signature morphism has a left adjoint. With some more effort, one can generalise this result and prove that in the standard equational institution the reduct functor induced by a *theory* morphism has a left adjoint:

Exercise 4.5.12. Prove that in the equational institution **EQ**, for any theory morphism $\sigma\!:\!T \to T'$, the reduct functor $\_\big|_\sigma\!:\!\mathbf{Mod}[T'] \to \mathbf{Mod}[T]$ has a left adjoint.

HINT: Formalise and complete the following construction: Let $T = \langle \Sigma, \Phi \rangle$ and $T' = \langle \Sigma', \Phi' \rangle$. For any Σ-algebra $A \in Mod[T]$, let $\Sigma(A)$ be its diagram signature, and let



be a pushout in the category of signatures. Then, let $\Delta^+(A) \subseteq \mathbf{Sen}_{\mathbf{EQ}}(\Sigma(A))$ be the positive diagram of A. Consider the presentation $\langle \Sigma'(A), \sigma'(\Delta^+(A)) \cup \iota'(\Phi') \rangle$. By Theorem 2.5.14, this has an initial model. Its ι'-reduct is a free object over A. (See also Exercise 3.5.11 for a slightly different line of reasoning.) □

We will come back to a careful, more abstract analysis of this construction later (cf. Theorem 4.5.18 below). For now, just notice that the construction not only uses the fact that the equational institution admits the method of diagrams, but also relies (directly or indirectly) on a number of simple facts about the reachability structure of the equational institution. We capture some of these additional properties in the following abstract definition:

Definition 4.5.13 (Abstract algebraic institution). An *abstract algebraic institution* is a finitely exact institution $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ with reachability structure that admits the method of diagrams, for which the following conditions hold:

- For any signature $\Sigma \in |\mathbf{Sign}|$, the category $\mathbf{Mod}(\Sigma)$ has all products (of sets of models) and is $\mathbf{E}_\Sigma$-co-well-powered (Definition 3.3.10).
- For any signature morphism $\sigma: \Sigma \to \Sigma'$, the σ-reduct functor preserves submodels (i.e., for all $m' \in \mathbf{M}_{\Sigma'}$, $m'|_\sigma \in \mathbf{M}_\Sigma$) and products.
- (*Abstraction condition*) For any signature Σ and Σ-models $M, N \in |\mathbf{Mod}(\Sigma)|$, if M and N are isomorphic then they satisfy exactly the same Σ-sentences. □

Example 4.5.14. The institutions **EQ** of equational logic, **PEQ** of partial equational logic, and **CEQ** of equational logic for continuous algebras are abstract algebraic institutions. □

Exercise 4.5.15. There is a certain asymmetry in the above definition: reduct functors in abstract algebraic institutions are required to preserve submodels but are not required to preserve quotients. Prove that in **EQ**, reduct functors preserve quotients as well: for all $\sigma: \Sigma \to \Sigma'$ and $e' \in \mathbf{E}_{\Sigma'}$, $e'|_\sigma \in \mathbf{E}_\Sigma$. Show, however, that this is not true in general in **PEQ**. □

4.5.3 Liberal abstract algebraic institutions

In Section 4.3 we have shown that it is possible to restrict attention to initial models of specifications written in an arbitrary institution, even if theories in the institution

are not guaranteed to have initial models in general. Similarly, data constraints make sense in an arbitrary institution even if reduct functors induced by theory morphisms are not guaranteed to have left adjoints. This flexibility is useful, but nevertheless it may be important to know whether or not a theory used in an initiality constraint has an initial model, or whether a theory morphism used in a data constraint has a corresponding free functor. In some institutions this is always the case: the equational institution **EQ** is one example (cf. Theorem 2.5.14 and Exercise 4.5.12). In the rest of this section we present a characterisation of institutions that have this property. Of course, very little can be done in the framework of an arbitrary institution: however, abstract algebraic institutions as introduced above provide a sufficiently rich background.

Definition 4.5.16 (Liberal institution). An institution **INS** *admits initial models* if every theory in **INS** has an initial model. **INS** is *liberal* if for every theory morphism $\sigma: T \to T'$ in **INS**, the σ-reduct functor $\_|_\sigma: \mathbf{Mod}[T'] \to \mathbf{Mod}[T]$ has a left adjoint.

Then, an abstract algebraic institution **INS** *admits reachable initial models* if every theory in **INS** has an initial model which is reachable. **INS** is *strongly liberal* if for every theory morphism $\sigma: T \to T'$ in **INS**, the σ-reduct functor $\_|_\sigma: \mathbf{Mod}[T'] \to \mathbf{Mod}[T]$ has a left adjoint $\mathbf{F}_\sigma: \mathbf{Mod}[T] \to \mathbf{Mod}[T']$ such that for any $M \in Mod[T]$, $\mathbf{F}_\sigma(M) \in Mod[T']$ is σ-reachable. □

In the last part of the definition we have slightly abused notation by using σ as both a *theory* morphism and a *signature* morphism (which in fact it is). It is important that the notion of σ-reachability used here is taken w.r.t. signature morphisms (cf. Definition 4.5.1) without taking into account the theory context.

Exercise 4.5.17. Find an institution that admits initial models but does not admit reachable initial models. HINT: Consider an algebraic signature Σ with a unary operation symbol $f: s \to s$. Show that the class of Σ-algebras satisfying the axiom $\exists! x{:}s \bullet f(x) = x$ has an initial model which is not reachable, where $\exists!$ reads "there exists a unique", that is, $\exists! x{:}s \bullet f(x) = x$ stands for $\exists x{:}s \bullet f(x) = x \wedge \forall x_1, x_2{:}s \bullet f(x_1) = x_1 \wedge f(x_2) = x_2 \Rightarrow x_1 = x_2$. □

For abstract algebraic institutions, the requirements introduced in Definition 4.5.16 are pairwise equivalent.

Theorem 4.5.18. *Let* **INS** *be an abstract algebraic institution.* **INS** *is liberal if and only if it admits initial models.*

Proof.

($\Rightarrow$): Let $T = \langle \Sigma, \Phi \rangle$ be a theory. Let $\iota_\Sigma: \Sigma_\varnothing \to \Sigma$ be the only signature morphism from the initial signature $\Sigma_\varnothing$ to Σ. Then $\iota_\Sigma: T_\varnothing \to T$ is a theory morphism, where $T_\varnothing = \langle \Sigma_\varnothing, Cl_{\Sigma_\varnothing}(\varnothing) \rangle$ is the initial theory, and so the reduct functor $\_|_{\iota_\Sigma}: \mathbf{Mod}[T] \to \mathbf{Mod}[T_\varnothing]$ has a left adjoint $\mathbf{F}_{\iota_\Sigma}: \mathbf{Mod}[T_\varnothing] \to \mathbf{Mod}[T]$. Now, there is exactly one $\Sigma_\varnothing$-model, say $M_\varnothing \in |\mathbf{Mod}[T_\varnothing]|$, and moreover, $\mathbf{F}_{\iota_\Sigma}(M_\varnothing)$ is an initial model of T.

($\Leftarrow$): We follow the proof for the equational institution **EQ** sketched in Exercise 4.5.12. For any theory morphism $\sigma: T \to T'$, where $T = \langle \Sigma, \Phi \rangle$ and $T' = \langle \Sigma', \Phi' \rangle$, and model $M \in Mod[T]$, we construct a model $\mathbf{F}_\sigma(M) \in Mod[T']$ with unit $\eta_M: M \to \mathbf{F}_\sigma(M)|_\sigma$ that is free over M w.r.t. $\_|_\sigma: \mathbf{Mod}[T'] \to \mathbf{Mod}[T]$.

Let $\Sigma(M)$ be the diagram signature for M with signature inclusion $\iota: \Sigma \hookrightarrow \Sigma(M)$, and let

$$\begin{array}{ccc} \Sigma(M) & \xrightarrow{\sigma'} & \Sigma'(M) \\ {\scriptstyle\iota}\uparrow & & \uparrow{\scriptstyle\iota'} \\ \Sigma & \xrightarrow[\sigma]{} & \Sigma' \end{array}$$

be a pushout in the category of signatures. Then, let $\Delta^+(M) \subseteq \mathbf{Sen}(\Sigma(M))$ be the positive diagram of M. Consider the presentation $\langle \Sigma'(M), \sigma'(\Delta^+(M)) \cup \iota'(\Phi') \rangle$. By the assumption, it has an initial model, say I. Put $\mathbf{F}_\sigma(M) = I|_{\iota'}$. Then, since by the satisfaction condition $I|_{\sigma'} \models_{\Sigma(M)} \Delta^+(M)$, $I|_{\sigma'} \in Ext(E(M))$ (cf. Exercise 4.5.11). Hence, there exists a (unique, since $E(M)$ is reachable) $\Sigma(M)$-model morphism $\widehat{\eta_M}: E(M) \to I|_{\sigma'}$. Put $\eta_M = \widehat{\eta_M}|_\iota: M \to \mathbf{F}_\sigma(M)|_\sigma$.

First, notice that since $I \models_{\Sigma'(M)} \iota'(\Phi')$, $\mathbf{F}_\sigma(M) \in Mod[T']$. Then, consider an arbitrary model $N \in Mod[T']$ and a Σ-model morphism $f: M \to N|_\sigma$.

By the definition of the diagram signature for M, $N|_\sigma$ has a unique ι-expansion to a $\Sigma(M)$-model $E_f(N|_\sigma)$ such that there exists a $\Sigma(M)$-model morphism $E(f): E(M) \to E_f(N|_\sigma)$ with $E(f)|_\iota = f$. Amalgamation yields a unique $\Sigma'(M)$-model $E^\sigma_f(N|_\sigma) \in |\mathbf{Mod}(\Sigma'(M))|$ with $E^\sigma_f(N|_\sigma)|_{\sigma'} = E_f(N|_\sigma)$ and $E^\sigma_f(N|_\sigma)|_{\iota'} = N$. Since $N \models_{\Sigma'} \Phi'$, $E^\sigma_f(N|_\sigma) \models_{\Sigma'(M)} \iota'(\Phi')$. Then, since $E_f(N|_\sigma) \in Ext(E(M))$, $E_f(N|_\sigma) \models_{\Sigma(M)} \Delta^+(M)$, and so $E^\sigma_f(N|_\sigma) \models_{\Sigma'(M)} \sigma'(\Delta^+(M))$. Consequently, we get a unique $\Sigma'(M)$-model morphism $\widehat{f'}: I \to E^\sigma_f(N|_\sigma)$. Put $f' = \widehat{f'}|_{\iota'}: \mathbf{F}_\sigma(M) \to N$. Notice that $\widehat{\eta_M}; \widehat{f'}|_{\sigma'}: E(M) \to E_f(N|_\sigma)$. Hence, since $E(M)$ is reachable, $\widehat{\eta_M}; \widehat{f'}|_{\sigma'} = E(f)$, and so we obtain $\eta_M; f'|_\sigma = f$. Moreover, f' is the only morphism with this property. To see this, suppose that for some $f'': \mathbf{F}_\sigma(M) \to N$, $\eta_M; f''|_\sigma = f$. Then, by the amalgamation property (this time for model morphisms) there exists a $\Sigma'(M)$-model morphism $\widehat{f''}: I \to E^\sigma_f(N|_\sigma)$ such that $\widehat{f''}|_{\iota'} = f''$ (and $\widehat{f''}|_{\sigma'} = E(f''|_\sigma): I|_{\sigma'} \to E_f(N|_\sigma)$). By initiality of I, $\widehat{f''} = \widehat{f'}$, and so $f'' = f'$, which completes the proof. □

Theorem 4.5.19. *Let* **INS** *be an abstract algebraic institution.* **INS** *is strongly liberal if and only if it admits reachable initial models.*

Proof. We extend the proof of the previous theorem, relying on the notation introduced there.

($\Rightarrow$): The only additional remark needed is that $\mathbf{F}_{\iota_\Sigma}(M_\varnothing)$ is reachable if it is ι_Σ-reachable (cf. Exercise 4.5.3).

($\Leftarrow$): We have to additionally prove that $\mathbf{F}_\sigma(M) = I\big|_{\iota'}$ is σ-reachable whenever I is reachable. To see this, consider an arbitrary submodel of $I\big|_{\iota'}$ with an isomorphic σ-reduct, say $m: N \to I\big|_{\iota'}$, where $m \in \mathbf{M}_{\Sigma'}$ and $m\big|_\sigma: N\big|_\sigma \to I\big|_{\sigma;\iota'}$ is an isomorphism. Put $f = \eta_M;(m\big|_\sigma)^{-1}: M \to N\big|_\sigma$. Then $f;m\big|_\sigma = \eta_M$, and so $m\big|_\sigma$ has an expansion to a $\Sigma(M)$-model morphism $E(m\big|_\sigma): E_f(N\big|_\sigma) \to E_{\eta_M}(I\big|_{\sigma;\iota'}) = I\big|_{\sigma'}$. Then, as in the corresponding part of the proof of Theorem 4.5.18, we get a unique $\Sigma'(M)$-model $E^\sigma_f(N\big|_\sigma) \in |\mathbf{Mod}(\Sigma'(M))|$ such that $E^\sigma_f(N\big|_\sigma)\big|_{\sigma'} = E_f(N\big|_\sigma)$ and $E^\sigma_f(N\big|_\sigma)\big|_{\iota'} = N$, and a $\Sigma'(M)$-model morphism $\widehat{f'}: I \to E^\sigma_f(N\big|_\sigma)$. On the other hand, by the amalgamation property again, there exists a unique $\Sigma'(M)$-model morphism $\widehat{m}: E^\sigma_f(N\big|_\sigma) \to I$ such that $\widehat{m}\big|_{\sigma'} = E(m\big|_\sigma)$ and $\widehat{m}\big|_{\iota'} = m$. By the initiality of I, $\widehat{f'};\widehat{m}$ is the identity, and so is $(\widehat{f'};\widehat{m})\big|_{\iota'} = \widehat{f'}\big|_{\iota'};m$. Thus, by Exercise 3.3.5, m is an isomorphism — which completes the proof. □

4.5.4 Characterising abstract algebraic institutions that admit reachable initial models

From the very beginning of work on algebraic specifications it has been known that the standard equational institution **EQ** admits reachable initial models (cf. Theorem 2.5.14). Moreover, the proof of this property generalises readily to the situation where conditional equations (even with infinite sets of premises) are permitted as axioms. On the other hand, Example 2.7.11 shows that if disjunction is permitted, the property is lost. Indeed, in the standard algebraic framework the infinitary conditional axioms, which define all non-empty quasi-varieties, form in some sense a borderline beyond which one cannot be sure of the existence of reachable initial models. We generalise this result to the framework of abstract algebraic institutions.

Theorem 4.5.20. *Let* **INS** *be an abstract algebraic institution.* **INS** *admits reachable initial models if and only if every class of models definable in* **INS** *is closed under products (of sets of models) and under submodels.*

Proof.

($\Leftarrow$): This follows directly by Lemma 3.3.12; just notice that any class of models closed under products and submodels is a *non-empty* quasi-variety (cf. Definition 3.3.11).

($\Rightarrow$): Let $\langle \Sigma, \Phi \rangle$ be a presentation in **INS**. We show the required closure properties of $Mod_\Sigma(\Phi)$.

(*Submodels*): Consider a model $M \in Mod_\Sigma(\Phi)$ and its submodel $m: N \to M$, $m \in \mathbf{M}_\Sigma$. Let $\Sigma(N)$ be a diagram signature for N with signature inclusion $\iota: \Sigma \to \Sigma(N)$, and let $\Delta^+(N) \subseteq \mathbf{Sen}(\Sigma(N))$ be the positive diagram of N. Recall that $Mod_{\Sigma(N)}(\Delta^+(N)) = Ext(E(N))$, where $E(N) \in \mathbf{Mod}(\Sigma(N))$ is the

diagram expansion of N. The presentation $\langle \Sigma(N), \Delta^+(N) \cup \iota(\Phi)\rangle$ has a reachable initial model, say I. We show that $I\big|_\iota$ is isomorphic to N, which in particular implies $N \in Mod_\Sigma(\Phi)$.

Since $I \models_{\Sigma(N)} \Delta^+(N)$, there exists a $\Sigma(N)$-model morphism $f\!:\!E(N) \to I$. Moreover, since I is reachable, $f \in \mathbf{E}_{\Sigma(N)}$ (by Theorem 3.3.8(4)) and hence also $f\big|_\iota \in \mathbf{E}_\Sigma$. Then, let $E_m(M)$ be the unique expansion of M to a $\Sigma(N)$-model with $E(m)\!:\!E(N) \to E_m(M)$ such that $E(m)\big|_\iota = m$. Since $M \models \Phi$, $E_m(M) \models_{\Sigma(N)} \iota(\Phi)$, and, since $E_m(M) \in Ext(E(N))$, $E_m(M) \models_{\Sigma(N)} \Delta^+(N)$. Hence, there is a (unique) morphism $g\!:\!I \to E_m(M)$. Now, since $E(N)$ is reachable, there exists at most one morphism from $E(N)$ to $E_m(M)$, and so we have $f;g = E(m)$, which implies $f\big|_\iota;g\big|_\iota = m \in \mathbf{M}_\Sigma$. Since $f\big|_\iota \in \mathbf{E}_\Sigma$, it follows from Exercise 3.3.5 that $f\big|_\iota\!:\!N \to I\big|_\iota$ is indeed an isomorphism.

(*Products*): Consider any family $M_i \in Mod_\Sigma(\Phi)$, $i \in J$, where J is any set (of indices). Let N with projections $\pi_i\!:\!N \to M_i$, $i \in J$, be the product of $\langle M_i\rangle_{i\in J}$. We proceed similarly as in the previous case: let $\Sigma(N)$ be a diagram signature for N with signature inclusion $\iota\!:\!\Sigma \to \Sigma(N)$, and let $\Delta^+(N) \subseteq \mathbf{Sen}(\Sigma(N))$ be the positive diagram of N. The presentation $\langle \Sigma(N), \Delta^+(N) \cup \iota(\Phi)\rangle$ has a reachable initial model, say I. We show that $I\big|_\iota$ is isomorphic to N, which implies that $N \in Mod_\Sigma(\Phi)$.

Just as in the previous case, there exists $f\!:\!E(N) \to I$ with $f\big|_\iota \in \mathbf{E}_\Sigma$.

Then, for $i \in J$, let $E_{\pi_i}(M_i)$ be the unique $\Sigma(N)$-model such that there is an expansion of π_i to a $\Sigma(N)$-model morphism $E(\pi_i)\!:\!E(N) \to E_{\pi_i}(M_i)$. $E_{\pi_i}(M_i)$ satisfies both $\Delta^+(N)$ and $\iota(\Phi)$, and so there exists a morphism $h_i\!:\!I \to E_{\pi_i}(M_i)$. Hence, by the definition of a product, there exists a (unique) Σ-model morphism $g\!:\!I\big|_\iota \to N$ such that for $i \in J$, $h_i\big|_\iota = g;\pi_i$. Moreover, for $i \in J$, since $E(N)$ is reachable and so there is at most one morphism from $E(N)$ to $E_{\pi_i}(M_i)$, $f;h_i = E(\pi_i)$. Consequently, $(f\big|_\iota;g);\pi_i = f\big|_\iota;h_i\big|_\iota = (f;h_i)\big|_\iota = E(\pi_i)\big|_\iota = \pi_i$. It follows that $f\big|_\iota;g$ is an isomorphism, and thus $f\big|_\iota \in \mathbf{E}_\Sigma$ implies that $f\big|_\iota\!:\!N \to I\big|_\iota$ is an isomorphism as well. □

Exercise 4.5.21. As we have mentioned earlier, institutions of single-sorted logics, like those in Exercises 4.4.10 and 4.4.16, are only semi-exact, rather than finitely exact.

Call an institution **INS** *almost abstract algebraic* if it satisfies all the assumptions imposed on abstract algebraic institution except for the requirement of finite exactness, instead of which we require that:

- **INS** is semi-exact; and
- for each signature $\Sigma \in |\mathbf{Sign}_{\mathbf{INS}}|$, the category $\mathbf{Mod}_{\mathbf{INS}}(\Sigma)$ of Σ-models has an initial object.

The above characterisation theorems nearly hold for almost abstract algebraic institutions:

- By direct inspection of their proofs, check that Theorem 4.5.20 as well as the "if" parts of Theorems 4.5.18 and 4.5.19 hold for almost abstract algebraic institutions.
- Prove that the "only if" part of Theorem 4.5.18 holds for almost abstract algebraic institutions. HINT: To show that a Σ-theory T has an initial model, consider the identity signature morphism as a morphism from the empty Σ-theory to T. Then use Exercise 3.5.17.
- Show that the "only if" part of Theorem 4.5.19 does not hold for almost abstract algebraic institutions. HINT: In **SSEQ**, the requirement of σ-reachability is trivial for any signature morphism σ. Consider the extension of **SSEQ** by sentences involving the quantifier "there exists a unique". □

4.6 Bibliographical remarks

This chapter has its origins in the seminal work of Goguen and Burstall on institutions. The reader may have noticed that the main paper on institutions [GB92] appeared later than many of its applications. The first appearance of institutions was in the semantics of Clear [BG80], under the name "language", and early versions of [GB92] were widely circulated, with [GB84a] as an early published version. Most of our terminology (signature, sentence, model, liberal institution, etc.) comes from [GB92]. There is a minor technical difference with respect to the definition given in [GB92]: we take the contravariant functor $\mathbf{Mod}_{\mathbf{INS}}$ to be $\mathbf{Mod}_{\mathbf{INS}}\colon \mathbf{Sign}_{\mathbf{INS}}^{op} \to \mathbf{Cat}$ rather than $\mathbf{Mod}_{\mathbf{INS}}\colon \mathbf{Sign}_{\mathbf{INS}} \to \mathbf{Cat}^{op}$. This is consistent with the further refinement of this definition in Chapter 10 as well as with the notion of an indexed category (cf. Section 3.4.3 and [TBG91]).

A large number of variants, generalisations and extensions of the notion of institution have been considered. In some work where model morphisms are not important, institutions were considered with classes (rather than categories) of models, e.g. [BG80]. Somewhat dually, one way to bring deduction into the realm of institutions is by considering categories (rather than sets) of sentences, where morphisms capture proofs. These variants were present in some unpublished versions of [GB92]; see also [MGDT07] for some elaboration on these possibilities.

One line of generalisation is to allow a space of truth values other than just the standard two-valued set, leading to proposals like galleries [May85] or generalised institutions [GB86]. General logics [Mes89] add an explicit notion of entailment and proof to institutions, see Chapter 9 for developments in this direction. Foundations [Poi88] include a similar idea, in addition imposing a rich indexed category structure on sentences. Context institutions [Paw96] offer an explicit notion of context and hence of open formulae and valuation as a part of the institution structure. There have also been attempts to relax the satisfaction condition, with for instance pre-institutions [SS93], [SS96], where the equivalence in the satisfaction conditions is split into two separately-imposed implications. This captures logical systems in which one or both of the directions of the satisfaction condition fail, as discussed

before Exercise 4.1.2. This applies to the so-called ultra-loose approach to algebraic specification [WB89], Extended ML [KST97] and various notions of behavioural satisfaction, see Chapter 8. (In [Gog91a], the satisfaction condition is satisfied for behavioural satisfaction but at the cost of restricting the notion of signature morphism.) Overall though, in spite of all these proposed variants and generalisations, most research has been based on the original notion, as we present it here.

The theory of institutions adopts a primarily model-theoretic view of logical systems. This does not preclude proof-theoretic investigation, see Chapter 9, but it does exclude logical systems that are inherently not based on the Tarskian notion of satisfaction of a sentence in a model. Typically such systems are centred around a notion of logical consequence that is defined via deduction, in contrast to our Definition 4.2.5. One such example would be non-monotonic logics [MT93], where increasing the set of premises can render consequences invalid. Other examples include substructural logics such as linear logic [Gir87], where changing the number of occurrences of premises, or their order, may affect deduction and change the set of valid consequences. Clearly, such logics cannot be directly represented as institutions, but see for instance [CM97] which indicates how an institution for linear logic can be defined by taking linear logic sequents (statements about consequence) as individual sentences. A view of logic based on proof rules and deduction underlies so-called "general logical frameworks", with Edinburgh LF [HHP93] as a prime example. For proposals in this direction related to institutions, see π-institutions [FS88] and also entailment systems [Mes89], [HST94], which re-emerge in Definition 9.1.2 below.

Sections 4.1.1 gives only the beginning of the long list of examples of logical systems that have been formalised as institutions. Standard examples of institutions (**EQ**, **FOP**, **Horn**, **Horn** without equality, $\mathbf{EQ}^{\Rightarrow}$) were in [GB92] with further standard algebraic variants in [Mos96b], and **CEQ** is from [Tar86b].

Dozens of other logical systems have been formalised as institutions. Some examples: [Bor00] defines an institution of higher-order logic based on HOL; [SML05] defines an institution with type class polymorphism; [Roş94] defines an institution of order-sorted equational logic; [ACEGG91] defines a family of institutions of multiple-valued logics, including logical systems arising from fuzzy set theory; [Dia00] defines an institution of constraint logic; [Cîr02] defines an institution with models that have both coalgebraic and algebraic components, and sentences involving modal formulae; [FC96] defines an institution of temporal logic; [LS00] defines an institution of hybrid systems based on the specification language of HyTech [HHWT97]; and [BH06a] defines the Col constructor-based observational logic institution based on viewing reachability and observability as dual concepts. The semantics of basic specifications in CASL [ST04] defines an institution, the rest of the semantics being defined in an institution-independent fashion. Alternatives to the standard CASL institution include: the institution underlying CoCASL, which includes cogeneration constraints, cofreeness constraints, and modal formulae [MSRR06]; the institution underlying HasCASL, with partial higher order functions, higher-order subtyping, shallow polymorphism, and type classes, designed for specifying functional programs [SM09]; an institution of labelled tran-

sition logic for specifying dynamic reactive systems [RAC99]; and the institution underlying CSP-CASL for describing systems of processes [Rog06]. The eight institutions involved in CafeOBJ [DF98] are defined in [DF02], with their combination leading to an institution via a version of the Grothendieck construction (Definition 3.4.58) that is applicable here [Dia02], and the Maude language [CDE$^+$02] is based on rewriting logic [Mes92] and on the institution of membership equational logic [Mes98] (with some technical nuances of their relationship pointed at in [CMRM10]). Institutions for three different UML diagram types are defined in [CK08a, CK08b, CK08c], with the relationships between them given by institution comorphisms (see Section 10.4 below). A spectrum of institutions capturing some aspects of Semantic Web languages are defined and linked with each other in [LLD06]. Different approaches to the specification of objects have led to the definition of a number of institutions, including [SCS94] which defines an institution of temporal logic for specifying object behaviour, [GD94b] which argues that an institution based on hidden-sorted algebra is relevant, and [Zuc99] which shows how to construct an institution with features for specifying dynamic aspects of systems using so-called "d-oids" from an institution for specifying static data. Finally, some slightly non-standard examples include two institutions for graph colouring in [Sco04], a way of viewing a database as an institution [Gog10], and a framework based on institutions for typed object-oriented, XML and other data models [Ala02].

Some of the examples of constructions on institutions in Section 4.1.2 were independently introduced by others. For instance, [Mes89] constructs an institution "out of thin air" starting with theories in an entailment system, the idea of which is presented in Examples 4.1.36 and 4.1.40. Incidentally, a very interesting exercise is to use the method of diagrams (Definition 4.5.9) to show how the construction of models from theories recovers the institution for which the entailment system that generates the theories was built.

Overall though, Section 4.1.2 only hints at the issue of how institutions should be defined. In particular, we do not discuss here the notion of a *parchment* [GB86], which offers one convenient way to present institutions in a concise and uniform style, at the same time ensuring that the satisfaction condition holds. See also [MTP97, MTP98] for variants of this notion and its use for combining presentations of logical systems.

The idea of data constraints originates in [BG80], but has been independently introduced earlier by Reichel [Rei80], cf. [KR71]. Our treatment in Section 4.3 follows [GB92]. Definition 4.3.8 is essentially equivalent to the definition there, although the technicalities are somewhat different; in particular, as in [ST88a], we do not require the institution to be liberal. Hierarchy constraints [SW82], also known as generating constraints [EWT83], are like data constraints but require that some carriers are generated from other carriers rather than freeness, see Exercise 4.3.13. Exercise 4.3.14 introduces a way to specify so-called co-inductive data types involving infinitary data. This has been mixed with algebraic techniques both in specification, see COCASL [MSRR06] and in experimental programming languages, see [Hag87] and Charity [CS92, CF92]. See [Rut00] for an introduction to a comprehen-

sive coalgebraic approach to specification which provides an alternative perspective to the material on behavioural specifications in Chapter 8 below.

Colimits of signatures and theories built over them have been used as a tool for combining theories and specifications at least since [BG77, GB78]. This follows the general ideas of [Gog73] and underlies for instance the semantics of Clear [BG80] and the commercial Specware system [Smi06]; support for the use of colimits to combine theories in a number of institutions is also offered by the HETS system [MML07]. A category-theoretic approach to software engineering which makes extensive use of these ideas is [Fia05]. Theorem 4.4.1 originates with [GB92], generalising a non-institutional version in [GB84b], and Corollary 4.4.2 is from [BG80].

The idea of amalgamation in model theory [CK90] refers to a subtler and deeper property of certain theories than does the notion defined here. The use of amalgamation in algebraic specification, in connection with pushout-style parameterisation mechanisms, originates with [EM85], following its introduction in [BPP85], see also the Extension Lemma in [EKT$^+$80, EKT$^+$83]. In the context of an arbitrary institution, it was first imposed as a requirement and linked with continuity of the model functor in [ST88a], cf. [EWT83].

Limiting the amalgamation property to pushouts along a chosen collection of signature morphisms, as in Definition 4.4.18, is important not only because of examples like those in Exercise 4.4.19. The range of relevant cases includes systems emerging in practice. For instance, the institution of CASL [Mos04] admits amalgamation for pushouts along most, but not all, CASL signature morphisms, due to problems with the required unique interpretation of subsorting coercions, see [SMT$^+$05].

There has been some confusion with the terminology surrounding exactness of institutions in the literature. The term was first used in [Mes89], although for preservation of signature pushouts (the amalgamation property) only. It became widely used after [DGS93], where it meant that the model functor maps finite colimits of signatures to limits in **Cat**, so that neither infinite colimits nor existence of colimits were covered (the latter also applies to semi-exactness as introduced there). This was sometimes missed in the literature, leading to subtle mistakes in the presentation of some results. We decided to put all of these assumptions together under the single requirement of "exactness". The notion of an institution "with composable signatures" was used in early versions of this chapter and in [Tar99] to mean the same thing as exactness, and this terminology was adopted by other authors in a few papers. The notion of exactness as used in category theory is different, although for functors between so-called Abelian categories it implies preservation of finite colimits.

The consequences of semi-exactness for preservation of finite connected colimits of signature diagrams stated in Proposition 4.4.15 appear to be new in the literature concerning institutions; they had not been clear to us until we were pointed to [CJ95] and a result there which we give as Exercise 3.4.55.

Institutions with extra structure have been used as the basis for the definition of the semantics of a number of specification languages, beginning with ASL [ST88a] which required an exact institution. In [ST86], an institution-independent semantics for the Extended ML specification language is sketched in terms of an "institution

with syntax"; this requires an additional functor which gives concrete syntactic representations of sentences, together with a natural transformation which associates these concrete objects with the "abstract" sentences they represent. In [ST04], the semantics of CASL is based on an "institution with qualified symbols" [Mos00] which requires considerable additional structure in order to support the operations on signatures used in the semantics; these include union of signatures and generation of signature morphisms from maps between symbols. Similar constructions on signatures are available when the category of signatures is equipped with a so-called inclusion system, which leads to the concept of an inclusive institution [DGS93], [GR04] (see also Exercise 5.2.1 below).

Although the theory of institutions emerged originally in the context of algebraic specification theory, it shares ideas and broad goals with abstract model theory as pursued within mathematical logic, see [Bar74, BF85], which concentrates on the study of definable classes of algebras (or rather first-order structures), abstracting away from the structure of sentences and from proof-theoretic mechanisms. The idea of developing an institutional version of abstract model theory, which also abstracts away from the nature of models, was first put forward in [Tar86a], where for instance the equivalence of the Craig interpolation and Robinson consistency properties, mentioned in Section 4.4.1, was shown.

The Craig interpolation property (Definition 4.4.21) will be used frequently in the sequel. In this formulation, it originates in [Tar86a]. Interpolation for first-order logic is a standard result in model theory [CK90] but the delicacy of its status in many-sorted first-order logic (see Exercise 4.4.23) was first pointed out in [Bor05]. There are several variants of the formulation of interpolation [DM00]. the generalisation to arbitrary commuting squares of signature morphisms [Dia08] and sets of interpolants (see the discussion in [DGS93]) is especially important. In particular, sets of interpolants may always be found in the case of equational logic under the assumption that carriers are non-empty [Rod91], but the necessity of this assumption has been widely disregarded, see Exercise 4.4.25.

Our treatment of variables, open formulae and quantification in an arbitrary institution comes from [Tar86b, ST88a]; see the concept of syntactic operator in [Bar74] for an earlier related idea. Section 4.5 is based on [Tar85], following [MM84] which is in an institutional style but based on the standard notion of logical structure. In [Tar86b], infinitary conditional "equations" were defined for an arbitrary abstract algebraic institution and it was shown that sets of these sentences define quasi-varieties, see [Mal71], thus obtaining a "syntactic" version of Theorem 4.5.20. Further developments in institutional abstract model theory, with results and ideas that refine those in Sections 4.4 and 4.5 and reach much further into classical model theory than we have done here, are in [Dia08].

References

AC89. Egidio Astesiano and Maura Cerioli. On the existence of initial models for partial (higher-order) conditional specifications. In Josep Díaz and Fernando Orejas, editors, *Proceedings of the International Joint Conference on Theory and Practice of Software Development, TAPSOFT'89*, Barcelona, *Lecture Notes in Computer Science*, volume 351, pages 74–88. Springer, 1989.

AC01. David Aspinall and Adriana B. Compagnoni. Subtyping dependent types. *Theoretical Computer Science*, 266(1–2):273–309, 2001.

ACEGG91. Jaume Agustí-Cullell, Francesc Esteva, Pere Garcia, and Lluis Godo. Formalizing multiple-valued logics as institutions. In Bernadette Bouchon-Meunier, Ronald R. Yager, and Lotfi A. Zadeh, editors, *Proceedings of the 3rd International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU'90*, Paris, *Lecture Notes in Computer Science*, volume 521, pages 269–278. Springer, 1991.

AF96. Mário Arrais and José Luiz Fiadeiro. Unifying theories in different institutions. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification. Selected Papers from the 11th Workshop on Specification of Abstract Data Types*, Oslo, *Lecture Notes in Computer Science*, volume 1130, pages 81–101. Springer, 1996.

AG97. Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, 1997.

AH05. David Aspinall and Martin Hofmann. Dependent types. In Benjamin Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 2, pages 45–86. MIT Press, 2005.

AHS90. Jiři Adámek, Horst Herrlich, and George Strecker. *Abstract and Concrete Categories: The Joy of Cats*. Wiley, 1990.

Ala02. Suad Alagic. Institutions: Integrating objects, XML and databases. *Information and Software Technology*, 44(4):207–216, 2002.

AM75. Michael A. Arbib and Ernest G. Manes. *Arrows, Structures and Functors: The Categorical Imperative*. Academic Press, 1975.

Asp95. David Aspinall. Subtyping with singleton types. In Leszek Pacholski and Jerzy Tiuryn, editors, *Proceedings of the 8th International Workshop on Computer Science Logic, CSL'94*, Kazimierz, *Lecture Notes in Computer Science*, volume 933, pages 1–15. Springer, 1995.

Asp97. David Aspinall. *Type Systems for Modular Programming and Specification*. PhD thesis, University of Edinburgh, Department of Computer Science, 1997.

Asp00. David Aspinall. Subtyping with power types. In Peter Clote and Helmut Schwichtenberg, editors, *Proceedings of the 14th International Workshop on Computer Science*

Logic, Fischbachau, *Lecture Notes in Computer Science*, volume 1862, pages 156–171. Springer, 2000.

Avr91. Arnon Avron. Simple consequence relations. *Information and Computation*, 92:105–139, 1991.

Awo06. Steve Awodey. *Category Theory*. Oxford University Press, 2006.

Bar74. Jon Barwise. Axioms for abstract model theory. *Annals of Mathematical Logic*, 7:221–265, 1974.

BBB+85. Friedrich L. Bauer, Rudolf Berghammer, Manfred Broy, Walter Dosch, Franz Geiselbrechtinger, Rupert Gnatz, E. Hangel, Wolfgang Hesse, Bernd Krieg-Brückner, Alfred Laut, Thomas Matzner, Bernd Möller, Friederike Nickl, Helmut Partsch, Peter Pepper, Klaus Samelson, Martin Wirsing, and Hans Wössner. *The Munich Project CIP: Volume 1: The Wide Spectrum Language CIP-L*, *Lecture Notes in Computer Science*, volume 183. Springer, 1985.

BBC86. Gilles Bernot, Michel Bidoit, and Christine Choppy. Abstract data types with exception handling: An initial approach based on a distinction between exceptions and errors. *Theoretical Computer Science*, 46(1):13–45, 1986.

BC88. Val Breazu-Tannen and Thierry Coquand. Extensional models for polymorphism. *Theoretical Computer Science*, 59(1–2):85–114, 1988.

BCH99. Michel Bidoit, María Victoria Cengarle, and Rolf Hennicker. Proof systems for structured specifications and their refinements. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 11, pages 385–433. Springer, 1999.

BD77. R.M. Burstall and J. Darlington. A transformational system for developing recursive programs. *Journal of the Association for Computing Machinery*, 24(1):44–67, 1977.

BDP+79. Manfred Broy, Walter Dosch, Helmut Partsch, Peter Pepper, and Martin Wirsing. Existential quantifiers in abstract data types. In Hermann A. Maurer, editor, *Proceedings of the 6th International Colloquium on Automata, Languages and Programming*, Graz, *Lecture Notes in Computer Science*, volume 71, pages 73–87. Springer, 1979.

Bén85. Jean Bénabou. Fibred categories and the foundations of naïve category theory. *Journal of Symbolic Logic*, 50:10–37, 1985.

Ber87. Gilles Bernot. Good functors ... are those preserving philosophy! In David H. Pitt, Axel Poigné, and David E. Rydeheard, editors, *Proceedings of the 2nd Summer Conference on Category Theory and Computer Science*, Edinburgh, *Lecture Notes in Computer Science*, volume 283, pages 182–195. Springer, 1987.

BF85. Jon Barwise and Solomon Feferman, editors. *Model-Theoretic Logics*. Springer, 1985.

BG77. R.M. Burstall and J.A. Goguen. Putting theories together to make specifications. In *Fifth International Joint Conference on Artificial Intelligence*, pages 1045–1058, Boston, 1977.

BG80. R.M. Burstall and J.A. Goguen. The semantics of Clear, a specification language. In Dines Bjørner, editor, *Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification*, *Lecture Notes in Computer Science*, volume 86, pages 292–332. Springer, 1980.

BG81. R.M. Burstall and J.A. Goguen. An informal introduction to specifications using Clear. In R.S. Boyer and J.S. Moore, editors, *The Correctness Problem in Computer Science*, pages 185–213. Academic Press, 1981. Also in: *Software Specification Techniques* (eds. N. Gehani and A.D. McGettrick), Addison-Wesley, 1986.

BG01. Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

BH96. Michel Bidoit and Rolf Hennicker. Behavioural theories and the proof of behavioural properties. *Theoretical Computer Science*, 165(1):3–55, 1996.

BH98. Michel Bidoit and Rolf Hennicker. Modular correctness proofs of behavioural implementations. *Acta Informatica*, 35(11):951–1005, 1998.

BH06a. Michel Bidoit and Rolf Hennicker. Constructor-based observational logic. *Journal of Logic and Algebraic Programming*, 67(1–2):3–51, 2006.

BH06b. Michel Bidoit and Rolf Hennicker. Proving behavioral refinements of COL-specifications. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning and Computation: Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 4060, pages 333–354. Springer, 2006.

BHK90. Jan Bergstra, Jan Heering, and Paul Klint. Module algebra. *Journal of the Association for Computing Machinery*, 37(2):335–372, 1990.

BHW94. Michel Bidoit, Rolf Hennicker, and Martin Wirsing. Characterizing behavioural semantics and abstractor semantics. In Donald Sannella, editor, *Proceedings of the 5th European Symposium on Programming*, Edinburgh, *Lecture Notes in Computer Science*, volume 788, pages 105–119. Springer, 1994.

BHW95. Michel Bidoit, Rolf Hennicker, and Martin Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25(2-3):149–186, 1995.

Bir35. Garrett Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.

BL69. R.M. Burstall and P.J. Landin. Programs and their proofs: an algebraic approach. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 17–43. Edinburgh University Press, 1969.

BM04. Michel Bidoit and Peter D. Mosses, editors. CASL *User Manual*. Number 2900 in Lecture Notes in Computer Science. Springer, 2004.

BN98. Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

Bor94. Francis Borceaux. *Handbook of Categorical Algebra*. Cambridge University Press, 1994.

Bor00. Tomasz Borzyszkowski. Higher-order logic and theorem proving for structured specifications. In Didier Bert, Christine Choppy, and Peter D. Mosses, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 14th International Workshop on Algebraic Development Techniques*, Château de Bonas, *Lecture Notes in Computer Science*, volume 1827, pages 401–418. Springer, 2000.

Bor02. Tomasz Borzyszkowski. Logical systems for structured specifications. *Theoretical Computer Science*, 286(2):197–245, 2002.

Bor05. Tomasz Borzyszkowski. Generalized interpolation in first order logic. *Fundamenta Informaticae*, 66(3):199–219, 2005.

BPP85. Edward K. Blum and Francesco Parisi-Presicce. The semantics of shared submodules specifications. In Hartmut Ehrig, Christiane Floyd, Maurice Nivat, and James W. Thatcher, editors, *Mathematical Foundations of Software Development. Proceedings of the International Joint Conference on Theory and Practice of Software Development. Volume 1: Colloquium on Trees in Algebra and Programming*, *Lecture Notes in Computer Science*, volume 185, pages 359–373. Springer, 1985.

BRJ98. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

BS93. Rudolf Berghammer and Gunther Schmidt. Relational specifications. In C. Rauszer, editor, *Proc. XXXVIII Banach Center Semester on Algebraic Methods in Logic and their Computer Science Applications*, *Banach Center Publications*, volume 28, pages 167–190, Warszawa, 1993. Institute of Mathematics, Polish Academy of Sciences.

BST02. Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Architectural specifications in CASL. *Formal Aspects of Computing*, 13:252–273, 2002.

BST08. Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Observational interpretation of CASL specifications. *Mathematical Structures in Computer Science*, 18:325–371, 2008.

BT87. Jan Bergstra and John Tucker. Algebraic specifications of computable and semicomputable data types. *Theoretical Computer Science*, 50(2):137–181, 1987.

BT96. Michel Bidoit and Andrzej Tarlecki. Behavioural satisfaction and equivalence in concrete model categories. In Hélène Kirchner, editor, *Proceedings of the 21st International Colloquium on Trees in Algebra and Programming*, Linköping, *Lecture Notes in Computer Science*, volume 1059, pages 241–256. Springer, 1996.

Bur86. Peter Burmeister. *A Model Theoretic Oriented Approach to Partial Algebras*. Akademie-Verlag, 1986.

BW82a. Friedrich L. Bauer and Hans Wössner. *Algorithmic Language and Program Development*. Springer, 1982.

BW82b. Manfred Broy and Martin Wirsing. Partial abstract data types. *Acta Informatica*, 18(1):47–64, 1982.

BW85. Michael Barr and Charles Wells. *Toposes, Triples and Theories*. Number 278 in Grundlehren der mathematischen Wissenschaften. Springer, 1985.

BW95. Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice Hall, second edition, 1995.

BWP84. Manfred Broy, Martin Wirsing, and Claude Pair. A systematic study of models of abstract data types. *Theoretical Computer Science*, 33(2–3):139–174, 1984.

Car88. Luca Cardelli. Structural subtyping and the notion of power type. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages*, San Diego, pages 70–79, 1988.

CDE+02. Manuel Clavela, 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*, 285(2):187–243, 2002. See also `http://maude.cs.uiuc.edu/`.

Cen94. María Victoria Cengarle. *Formal Specifications with Higher-Order Parameterization*. PhD thesis, Ludwig-Maximilians-Universität München, Institut für Informatik, 1994.

CF92. Robin Cockett and Tom Fukushima. About Charity. Technical Report No. 92/480/18, Department of Computer Science, University of Calgary, 1992.

CGR03. Carlos Caleiro, Paula Gouveia, and Jaime Ramos. Completeness results for fibred parchments: Beyond the propositional base. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 16th International Workshop on Algebraic Development Techniques*, Frauenchiemsee, *Lecture Notes in Computer Science*, volume 2755, pages 185–200. Springer, 2003.

Chu56. Alonzo Church. *Introduction to Mathematical Logic, Volume 1*. Princeton University Press, 1956.

Cîr02. Corina Cîrstea. On specification logics for algebra-coalgebra structures: Reconciling reachability and observability. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures. European Joint Conferences on Theory and Practice of Software (ETAPS 2002)*, Grenoble, *Lecture Notes in Computer Science*, volume 2303, pages 82–97. Springer, 2002.

CJ95. Aurelio Carboni and Peter Johnstone. Connected limits, familial representability and Artin glueing. *Mathematical Structures in Computer Science*, 5(4):441–459, 1995.

CK90. Chen-Chung Chang and H. Jerome Keisler. *Model Theory*. North-Holland, third edition, 1990.

CK08a. María Victoria Cengarle and Alexander Knapp. An institution for OCL 2.0. Technical Report I0801, Institut für Informatik, Ludwig-Maximilians-Universität München, 2008.

CK08b. María Victoria Cengarle and Alexander Knapp. An institution for UML 2.0 interactions. Technical Report I0808, Institut für Informatik, Ludwig-Maximilians-Universität München, 2008.

CK08c. María Victoria Cengarle and Alexander Knapp. An institution for UML 2.0 static structures. Technical Report I0807, Institut für Informatik, Ludwig-Maximilians-Universität München, 2008.

CKTW08. Maria-Victoria Cengarle, Alexander Knapp, Andrzej Tarlecki, and Martin Wirsing. A heterogeneous approach to UML semantics. In Pierpaolo Degano, Rocco de Nicola, and José Meseguer, editors, *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 5065, pages 383–402. Springer, 2008.

CM97. Maura Cerioli and José Meseguer. May I borrow your logic? (Transporting logical structures along maps). *Theoretical Computer Science*, 173(2):311–347, 1997.

CMRM10. Mihai Codescu, Till Mossakowski, Adrían Riesco, and Christian Maeder. Integrating Maude into Hets. In Mike Johnson and Dusko Pavlovic, editors, *AMAST 2010*, Lecture Notes in Computer Science. Springer, 2010.

CMRS01. Carlos Caleiro, Paulo Mateus, Jaime Ramos, and Amílcar Sernadas. Combining logics: Parchments revisited. In Maura Cerioli and Gianna Reggio, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 15th Workshop on Algebraic Development Techniques joint with the CoFI WG Meeting*, Genova, *Lecture Notes in Computer Science*, volume 2267, pages 48–70. Springer, 2001.

Coh65. Paul M. Cohn. *Universal Algebra*. Harper and Row, 1965.

CS92. Robin Cockett and Dwight Spencer. Strong categorical datatypes I. In R.A.G. Seely, editor, *International Meeting on Category Theory 1991*, Canadian Mathematical Society Proceedings. American Mathematical Society, 1992.

CSS05. Carlos Caleiro, Amílcar Sernadas, and Cristina Sernadas. Fibring logics: Past, present and future. In Sergei N. Artemov, Howard Barringer, Artur S. d'Avila Garcez, Luís C. Lamb, and John Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay, Volume One*, pages 363–388. College Publications, 2005.

DF98. Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, *AMAST Series in Computing*, volume 6. World Scientific, 1998.

DF02. Răzvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. *Theoretical Computer Science*, 285:289–318, 2002.

DGS93. Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gérard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge University Press, 1993.

Dia00. Răzvan Diaconescu. Category-based constraint logic. *Mathematical Structures in Computer Science*, 10(3):373–407, 2000.

Dia02. Răzvan Diaconescu. Grothendieck institutions. *Applied Categorical Structures*, 10(4):383–402, 2002.

Dia08. Răzvan Diaconescu. *Institution-independent Model Theory*. Birkhäuser, 2008.

DJ90. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B (Formal Models and Semantics)*, pages 244–320. North-Holland and MIT Press, 1990.

DLL62. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

DM00. Theodosis Dimitrakos and Tom Maibaum. On a generalised modularisation theorem. *Information Processing Letters*, 74(1–2):65–71, 2000.

DMR76. Martin Davis, Yuri Matiyasevich, and Julia Robinson. Hilbert's tenth problem. Diophantine equations: Positive aspects of a negative solution. In *Mathematical Developments Arising from Hilbert Problems*, *Proceedings of Symposia in Pure Mathematics*, volume 28, pages 323–378, Providence, Rhode Island, 1976. American Mathematical Society.

DP90. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

Ehr78. Hans-Dieter Ehrich. Extensions and implementations of abstract data type specifications. In Józef Winkowski, editor, *Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science*, Zakopane, *Lecture Notes in Computer Science*, volume 64, pages 155–164. Springer, 1978.

Ehr81. Hans-Dieter Ehrich. On realization and implementation. In Jozef Gruska and Michal Chytil, editors, *Proceedings of the 10th Symposium on Mathematical Foundations of Computer Science*, Štrbské Pleso, *Lecture Notes in Computer Science*, volume 118, pages 271–280. Springer, 1981.

Ehr82. Hans-Dieter Ehrich. On the theory of specification, implementation and parametrization of abstract data types. *Journal of the Association for Computing Machinery*, 29(1):206–227, 1982.

EKMP82. Hartmut Ehrig, Hans-Jörg Kreowski, Bernd Mahr, and Peter Padawitz. Algebraic implementation of abstract data types. *Theoretical Computer Science*, 20:209–263, 1982.

EKT+80. Hartmut Ehrig, Hans-Jörg Kreowski, James Thatcher, Eric Wagner, and Jesse Wright. Parameter passing in algebraic specification languages. Technical report, Technische Universität Berlin, 1980.

EKT+83. Hartmut Ehrig, Hans-Jörg Kreowski, James Thatcher, Eric Wagner, and Jesse Wright. Parameter passing in algebraic specification languages. *Theoretical Computer Science*, 28(1–2):45–81, 1983.

EM85. Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1*, *EATCS Monographs on Theoretical Computer Science*, volume 6. Springer, 1985.

Eme90. E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B (Formal Models and Semantics)*, pages 995–1072. North-Holland and MIT Press, 1990.

End72. Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

EPO89. Hartmut Ehrig, Peter Pepper, and Fernando Orejas. On recent trends in algebraic specification. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *Proceeding of the 16th International Colloquium on Automata, Languages and Programming*, Stresa, *Lecture Notes in Computer Science*, volume 372, pages 263–288. Springer, 1989.

EWT83. Hartmut Ehrig, Eric G. Wagner, and James W. Thatcher. Algebraic specifications with generating constraints. In *Proceeding of the 10th International Colloquium on Automata, Languages and Programming*, Barcelona, *Lecture Notes in Computer Science*, volume 154, pages 188–202. Springer, 1983.

Far89. Jordi Farrés-Casals. Proving correctness of constructor implementations. In Antoni Kreczmar and Grazyna Mirkowska, editors, *Proceedings of the 14th Symposium on Mathematical Foundations of Computer Science*, Porabka-Kozubnik, *Lecture Notes in Computer Science*, volume 379, pages 225–235. Springer, 1989.

Far90. Jordi Farrés-Casals. Proving correctness wrt specifications with hidden parts. In Hélène Kirchner and Wolfgang Wechler, editors, *Proceedings of the 2nd International Conference on Algebraic and Logic Programming*, Nancy, *Lecture Notes in Computer Science*, volume 463, pages 25–39. Springer, 1990.

Far92. Jordi Farrés-Casals. *Verification in ASL and Related Specification Languages*. PhD thesis, University of Edinburgh, Department of Computer Science, 1992.

FC96. José Luiz Fiadeiro and José Félix Costa. Mirror, mirror in my hand: A duality between specifications and models of process behaviour. *Mathematical Structures in Computer Science*, 6(4):353–373, 1996.

Fei89. Loe M. G. Feijs. The calculus $\lambda\pi$. In Martin Wirsing and Jan A. Bergstra, editors, *Proceedings of the Workshop on Algebraic Methods: Theory, Tools and Applications*, *Lecture Notes in Computer Science*, volume 394, pages 307–328. Springer, 1989.

FGT92. William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. Little theories. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, *Lecture Notes in Artificial Intelligence*, volume 607, pages 567–581, Saratoga Springs, 1992. Springer.

Fia05. José Luiz Fiadeiro. *Categories for Software Engineering*. Springer, 2005.

Fit08. John S. Fitzgerald. The typed logic of partial functions and the Vienna Development Method. In Dines Bjørner and Martin Henson, editors, *Logics of Specification Languages*, pages 453–487. Springer, 2008.

FJ90. J. Fitzgerald and C.B. Jones. Modularizing the formal description of a database system. In *Proceedings of the 3rd International Symposium of VDM Europe: VDM and Z, Formal Methods in Software Development*, Kiel, *Lecture Notes in Computer Science*, volume 428, pages 189–210. Springer, 1990.

FS88. José Luiz Fiadeiro and Amílcar Sernadas. Structuring theories on consequence. In Donald Sannella and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification. Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, *Lecture Notes in Computer Science*, volume 332, pages 44–72. Springer, 1988.

Gab98. Dov M. Gabbay. *Fibring Logics*, *Oxford Logic Guides*, volume 38. Oxford University Press, 1998.

Gan83. Harald Ganzinger. Parameterized specifications: Parameter passing and implementation with respect to observability. *ACM Transactions on Programming Languages and Systems*, 5(3):318–354, 1983.

GB78. J.A. Goguen and R.M. Burstall. Some fundamental properties of algebraic theories: a tool for semantics of computation. Technical Report 53, Department of Artificial Intelligence, University of Edinburgh, 1978. Revised version appeared as [GB84b] and [GB84c].

GB80. J.A. Goguen and R.M. Burstall. CAT, a system for the structured elaboration of correct programs from structured specifications. Technical Report CSL-118, Computer Science Laboratory, SRI International, 1980.

GB84a. J.A. Goguen and R.M. Burstall. Introducing institutions. In Edmund Clarke and Dexter Kozen, editors, *Proceedings of the Workshop on Logics of Programs*, Pittsburgh, *Lecture Notes in Computer Science*, volume 164, pages 221–256. Springer, 1984.

GB84b. J.A. Goguen and R.M. Burstall. Some fundamental algebraic tools for the semantics of computation. Part 1: Comma categories, colimits, signatures and theories. *Theoretical Computer Science*, 31:175–209, 1984.

GB84c. J.A. Goguen and R.M. Burstall. Some fundamental algebraic tools for the semantics of computation. Part 2: Signed and abstract theories. *Theoretical Computer Science*, 31:263–295, 1984.

GB86. Joseph A. Goguen and Rod M. Burstall. A study in the functions of programming methodology: Specifications, institutions, charters and parchments. In David H. Pitt, Samson Abramsky, Axel Poigné, and David E. Rydeheard, editors, *Proceedings of the Tutorial and Workshop on Category Theory and Computer Programming*, Guildford, *Lecture Notes in Computer Science*, volume 240, pages 313–333. Springer, 1986.

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

GD94a. Joseph Goguen and Răzvan Diaconescu. An Oxford survey of order sorted algebra. *Mathematical Structures in Computer Science*, 4(3):363–392, 1994.

GD94b. Joseph A. Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Hartmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification. Selected Papers from the 9th Workshop on Specification of Abstract Data Types joint with the 4th* COMPASS *Workshop*, Caldes de Malavella, *Lecture Notes in Computer Science*, volume 785, pages 1–29. Springer, 1994.

GDLE84. Martin Gogolla, Klaus Drosten, Udo Lipeck, and Hans-Dieter Ehrich. Algebraic and operational semantics of specifications allowing exceptions and errors. *Theoretical Computer Science*, 34(3):289–313, 1984.

GG89. Stephen J. Garland and John V. Guttag. An overview of LP, the Larch Prover. In *Third International Conference on Rewriting Techniques and Applications*, Chapel Hill, *Lecture Notes in Computer Science*, volume 355, pages 137–151. Springer, 1989. See also `http://nms.lcs.mit.edu/larch/LP/all.html`.

GGM76. V. Giarratana, F. Gimona, and Ugo Montanari. Observability concepts in abstract data type specifications. In Antoni Mazurkiewicz, editor, *Proceedings of the 5th Sympo-*

sium on Mathematical Foundations of Computer Science, Gdańsk, *Lecture Notes in Computer Science*, volume 45, pages 567–578. Springer, 1976.

GH78. John Guttag and James Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

GH93. John V. Guttag and James J. Horning. *Larch: Languages and Tools for Formal Specification*. Springer, 1993.

Gin68. Abraham Ginzburg. *Algebraic Theory of Automata*. Academic Press, 1968.

Gir87. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

Gir89. Jean-Yves Girard. *Proofs and Types*, *Cambridge Tracts in Theoretical Computer Science*, volume 7. Cambridge University Press, 1989. Translated and with appendices by Paul Taylor and Yves Lafont.

GLR00. Joseph Goguen, Kai Lin, and Grigore Roşu. Circular coinductive rewriting. In *Proceedings of the 15th International Conference on Automated Software Engineering*, Grenoble. IEEE Computer Society, 2000.

GM82. Joseph A. Goguen and José Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In Mogens Nielsen and Erik Meineche Schmidt, editors, *Proceeding of the 9th International Colloquium on Automata, Languages and Programming*, Aarhus, *Lecture Notes in Computer Science*, volume 140, pages 265–281. Springer, 1982.

GM85. Joseph Goguen and José Meseguer. Completeness of many sorted equational deduction. *Houston Journal of Mathematics*, 11(3):307–334, 1985.

GM92. Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.

GM00. Joseph A. Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000.

Gog73. Joseph Goguen. Categorical foundations for general systems theory. In F. Pichler and R. Trappl, editors, *Advances in Cybernetics and Systems Research*, London, pages 121–130. Transcripta Books, 1973.

Gog74. J.A. Goguen. Semantics of computation. In Ernest G. Manes, editor, *Proceedings of the 1st International Symposium on Category Theory Applied to Computation and Control*, San Francisco, *Lecture Notes in Computer Science*, volume 25, pages 151–163. Springer, 1974.

Gog78. Joseph Goguen. Abstract errors for abstract data types. In Erich Neuhold, editor, *Formal Description of Programming Concepts*, pages 491–526. North-Holland, 1978.

Gog84. Martin Gogolla. Partially ordered sorts in algebraic specifications. In *Proceedings of the 9th Colloquium on Trees in Algebra and Programming*, pages 139–153. Cambridge University Press, 1984.

Gog85. Martin Gogolla. A final algebra semantics for errors and exceptions. In Hans-Jörg Kreowski, editor, *Recent Trends in Data Type Specification. Selected Papers from the 3rd Workshop on Theory and Applications of Abstract Data Types*, Bremen, *Informatik-Fachberichte*, volume 116, pages 89–103. Springer, 1985.

Gog91a. Joseph Goguen. Types as theories. In G.M. Reed, A.W. Roscoe, and R.F. Wachter, editors, *Topology and Category Theory in Computer Science*, Oxford, pages 357–390. Oxford University Press, 1991.

Gog91b. Joseph A. Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67, 1991.

Gog96. Joseph A. Goguen. Parameterized programming and software architecture. In Murali Sitaraman, editor, *Proceedings of the Fourth International Conference on Software Reuse*, pages 2–11. IEEE Computer Society Press, 1996.

Gog10. Joseph Goguen. Information integration in institutions. In Larry Moss, editor, *Thinking Logically: a Volume in Memory of Jon Barwise*. CSLI, Stanford University, 2010. To appear.

Gol06. Robert Goldblatt. *Topoi: The Categorial Analysis of Logic*. Dover, revised edition, 2006.

Gor95. Andrew D. Gordon. Bisimilarity as a theory of functional programming. In *Proceedings of the 11th Annual Conference on Mathematical Foundations of Programming Semantics*. *Electronic Notes in Theoretical Computer Science*, 1:232–252, 1995.

GR02. Joseph A. Goguen and Grigore Roşu. Institution morphisms. *Formal Aspects of Computing*, 13(3-5):274–307, 2002.

GR04. Joseph A. Goguen and Grigore Roşu. Composing hidden information modules over inclusive institutions. In *From Object-Orientation to Formal Methods. Essays in Memory of Ole-Johan Dahl*, *Lecture Notes in Computer Science*, volume 2635, pages 96–123. Springer, 2004.

Grä79. George A. Grätzer. *Universal Algebra*. Springer, second edition, 1979.

GS90. Carl Gunter and Dana Scott. Semantic domains. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B (Formal Models and Semantics)*, pages 633–674. North-Holland and MIT Press, 1990.

GTW76. Joseph Goguen, James Thatcher, and Eric Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. Technical Report RC 6487, IBM Watson Research Center, Yorktown Heights NY, 1976. Also in: *Current Trends in Programming Methodology. Volume IV (Data Structuring)* (ed. R.T. Yeh), Prentice-Hall, 80–149, 1978.

GTWW73. Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. A junction between computer science and category theory, I: Basic concepts and examples (part 1). Technical Report RC 4526, IBM Watson Research Center, Yorktown Heights NY, 1973.

GTWW75. Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. An introduction to categories, algebraic theories and algebras. Technical Report RC 5369, IBM Watson Research Center, Yorktown Heights NY, 1975.

GTWW77. Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, 24(1):68–95, 1977.

Gut75. John Guttag. *The Specification and Application to Programming of Abstract Data Types*. PhD thesis, University of Toronto, Department of Computer Science, 1975.

Hag87. Tatsuya Hagino. *A Categorical Programming Language*. PhD thesis, University of Edinburgh, Department of Computer Science, 1987.

Häh01. Reiner Hähnle. Tableaux and related methods. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 100–178. Elsevier and MIT Press, 2001.

Hal70. Paul R. Halmos. *Naive Set Theory*. Undergraduate Texts in Mathematics. Springer, 1970.

Hat82. William Hatcher. *The Logical Foundations of Mathematics*. Foundations and Philosophy of Science and Technology. Pergamon Press, 1982.

Hay94. Susumu Hayashi. Singleton, union and intersection types for program extraction. *Information and Computation*, 109(1/2):174–210, 1994.

Hee86. Jan Heering. Partial evaluation and ω-completeness of algebraic specifications. *Theoretical Computer Science*, 43:149–167, 1986.

Hen91. Rolf Hennicker. Context induction: A proof principle for behavioural abstractions and algebraic implementations. *Formal Aspects of Computing*, 3(4):326–345, 1991.

HHP93. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

HHWT97. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

Hig63. Phillip J. Higgins. Algebras with a scheme of operators. *Mathematische Nachrichten*, 27:115–132, 1963.

HLST00. Furio Honsell, John Longley, Donald Sannella, and Andrzej Tarlecki. Constructive data refinement in typed lambda calculus. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures. European Joint Conferences on Theory and Practice of Software (ETAPS 2000)*, Berlin, *Lecture Notes in Computer Science*, volume 1784, pages 161–176. Springer, 2000.

Hoa72. C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

HS73. Horst Herrlich and George E. Strecker. *Category Theory: An Introduction*. Allyn and Bacon, 1973.

HS96. Martin Hofmann and Donald Sannella. On behavioural abstraction and behavioural satisfaction in higher-order logic. *Theoretical Computer Science*, 167:3–45, 1996.

HS02. Furio Honsell and Donald Sannella. Prelogical relations. *Information and Computation*, 178:23–43, 2002.

HST94. Robert Harper, Donald Sannella, and Andrzej Tarlecki. Structured presentations and logic representations. *Annals of Pure and Applied Logic*, 67:113–160, 1994.

Hus92. Heinrich Hussmann. Nondeterministic algebraic specifications and nonconfluent term rewriting. *Journal of Logic Programming*, 12(1–4):237–255, 1992.

HWB97. Rolf Hennicker, Martin Wirsing, and Michel Bidoit. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173(2):393–443, 1997.

Jac99. Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1999.

JL87. Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, Munich, pages 111–119, 1987.

JNW96. André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

JOE95. Rosa M. Jiménez, Fernando Orejas, and Hartmut Ehrig. Compositionality and compatibility of parameterization and parameter passing in specification languages. *Mathematical Structures in Computer Science*, 5(2):283–314, 1995.

Joh02. Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Oxford Logic Guides Series. Clarendon Press, 2002.

Jon80. Cliff B. Jones. *Software Development: A Rigorous Approach*. Prentice-Hall, 1980.

Jon89. Hans B.M. Jonkers. An introduction to COLD-K. In Martin Wirsing and Jan A. Bergstra, editors, *Proceedings of the Workshop on Algebraic Methods: Theory, Tools and Applications*, *Lecture Notes in Computer Science*, volume 394, pages 139–205. Springer, 1989.

JR97. Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Computer Science*, 62:222–259, 1997.

KB70. Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

Kir99. Hélène Kirchner. Term rewriting. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 9, pages 273–320. Springer, 1999.

KKM88. Claude Kirchner, Hélène Kirchner, and José Meseguer. Operational semantics of OBJ-3. In Timo Lepistö and Arto Salomaa, editors, *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, Tampere, *Lecture Notes in Computer Science*, volume 317, pages 287–301. Springer, 1988.

Klo92. Jan Klop. Term rewriting systems. In Samson Abramsky, Dov Gabbay, and Tom Maibaum, editors, *Handbook of Logic in Computer Science. Volume 2 (Background: Computational Structures)*, pages 1–116. Oxford University Press, 1992.

KM87. Deepak Kapur and David R. Musser. Proof by consistency. *Artificial Intelligence*, 31(2):125–157, 1987.

KR71. Heinz Kaphengst and Horst Reichel. Algebraische Algorithmentheorie. Technical Report WIB 1, VEB Robotron, Zentrum für Forschung und Technik, Dresden, 1971.

Kre87. Hans-Jörg Kreowski. Partial algebras flow from algebraic specifications. In T. Ottmann, editor, *Proceedings of the 14th International Colloquium on Automata, Languages and Programming*, Karlsruhe, *Lecture Notes in Computer Science*, volume 267, pages 521–530. Springer, 1987.

KST97. Stefan Kahrs, Donald Sannella, and Andrzej Tarlecki. The definition of Extended ML: A gentle introduction. *Theoretical Computer Science*, 173:445–484, 1997.

KTB91. Beata Konikowska, Andrzej Tarlecki, and Andrzej Blikle. A three-valued logic for software specification and validation. *Fundamenta Informaticae*, 14(4):411–453, 1991.

Las98. Sławomir Lasota. Open maps as a bridge between algebraic observational equivalence and bisimilarity. In Francesco Parisi-Presicce, editor, *Recent Trends in Data Type Specification. Selected Papers from the 12th International Workshop on Specification of Abstract Data Types*, Tarquinia, *Lecture Notes in Computer Science*, volume 1376, pages 285–299. Springer, 1998.

Law63. F. William Lawvere. *Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963.

LB88. Butler Lampson and Rod Burstall. Pebble, a kernel language for modules and abstract data types. *Information and Computation*, 76(2/3):278–346, 1988.

LEW96. Jacques Loeckx, Hans-Dieter Ehrich, and Markus Wolf. *Specification of Abstract Data Types*. John Wiley and Sons, 1996.

Lin03. Kai Lin. *Machine Support for Behavioral Algebraic Specification and Verification*. PhD thesis, University of California, San Diego, 2003.

Lip83. Udo Lipeck. *Ein algebraischer Kalkül für einen strukturierten Entwurf von Datenabstraktionen*. PhD thesis, Universität Dortmund, 1983.

LLD06. Dorel Lucanu, Yuan-Fang Li, and Jin Song Dong. Semantic Web languages—towards an institutional perspective. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning and Computation: Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 4060, pages 99–123. Springer, 2006.

LS86. Joachim Lambek and Philip J. Scott. *Introduction to Higher-Order Categorical Logic*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1986.

LS00. Hugo Lourenço and Amílcar Sernadas. An institution of hybrid systems. In Didier Bert, Christine Choppy, and Peter D. Mosses, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 14th International Workshop on Algebraic Development Techniques*, Château de Bonas, *Lecture Notes in Computer Science*, volume 1827, pages 219–236. Springer, 2000.

Luo93. Zhaohui Luo. Program specification and data refinement in type theory. *Mathematical Structures in Computer Science*, 3(3):333–363, 1993.

Mac71. Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

Mac84. David B. MacQueen. Modules for Standard ML. In *Proceedings of the 1984 ACM Conference on LISP and Functional Programming*, pages 198–207, 1984.

MAH06. Till Mossakowski, Serge Autexier, and Dieter Hutter. Development graphs — proof management for structured specifications. *Journal of Logic and Algebraic Programming*, 67(1–2):114–145, 2006.

Mai72. Tom Maibaum. The characterization of the derivation trees of context free sets of terms as regular sets. In *Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory*, pages 224–230, 1972.

Maj77. Mila E. Majster. Limits of the "algebraic" specification of abstract data types. *ACM SIGPLAN Notices*, 12(10):37–42, 1977.

Mal71. Anatoly Malcev. Quasiprimitive classes of abstract algebras in the metamathematics of algebraic systems. In *Mathematics of Algebraic Systems: Collected Papers, 1936-67*, number 66 in Studies in Logic and Mathematics, pages 27–31. North-Holland, 1971.

Man76. Ernest G. Manes. *Algebraic Theories*. Springer, 1976.

May85. Brian Mayoh. Galleries and institutions. Technical Report DAIMI PB-191, Aarhus University, 1985.

Mei92. Karl Meinke. Universal algebra in higher types. *Theoretical Computer Science*, 100:385–417, 1992.

Mes89. José Meseguer. General logics. In H.-D. Ebbinghaus, editor, *Logic Colloquium '87*, Granada, pages 275–329. North-Holland, 1989.

Mes92. José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

Mes98. José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi-Presicce, editor, *Recent Trends in Data Type Specification. Selected Papers from the 12th International Workshop on Specification of Abstract Data Types*, Tarquinia, *Lecture Notes in Computer Science*, volume 1376, pages 18–61. Springer, 1998.

Mes09. José Meseguer. Order-sorted parameterization and induction. In Jens Palsberg, editor, *Semantics and Algebraic Specification: Essays Dedicated to Peter D. Mosses on the Occasion of His 60th Birthday*, *Lecture Notes in Computer Science*, volume 5700, pages 43–80. Springer, 2009.

MG85. José Meseguer and Joseph Goguen. Initiality, induction and computability. In Maurice Nivat and John C. Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge, 1985.

MGDT07. Till Mossakowski, Joseph Goguen, Răzvan Diaconescu, and Andrzej Tarlecki. What is a logic? In Jean-Yves Beziau, editor, *Logica Universalis: Towards a General Theory of Logic*, pages 111–135. Birkhäuser, 2007.

MHST08. Till Mossakowski, Anne Haxthausen, Donald Sannella, and Andrzej Tarlecki. CASL — the common algebraic specification language. In Dines Bjørner and Martin Henson, editors, *Logics of Specification Languages*, pages 241–298. Springer, 2008.

Mid93. Aart Middeldorp. Modular properties of conditional term rewriting systems. *Information and Computation*, 104(1):110–158, 1993.

Mil71. Robin Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489, 1971.

Mil77. Robin Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4(1):1–22, 1977.

Mil89. Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

Mit96. John C. Mitchell. *Foundations of Programming Languages*. MIT Press, 1996.

MM84. Bernd Mahr and Johann Makowsky. Characterizing specification languages which admit initial semantics. *Theoretical Computer Science*, 31:49–60, 1984.

MML07. Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, HETS. In Orna Grumberg and Michael Huth, editors, *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. European Joint Conferences on Theory and Practice of Software (ETAPS 2007)*, Braga, *Lecture Notes in Computer Science*, volume 4424, pages 519–522. Springer, 2007. See also `http://www.informatik.uni-bremen.de/cofi/hets/`.

Mog91. Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

Moo56. Edward F. Moore. Gedanken-experiments on sequential machines. In Claude E. Shannon and John McCarthy, editors, *Annals of Mathematics Studies 34, Automata Studies*, pages 129–153. Princeton University Press, 1956.

Mos89. Peter D. Mosses. Unified algebras and modules. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*, Austin, pages 329–343, 1989.

Mos93. Peter Mosses. The use of sorts in algebraic specifications. In Michel Bidoit and Christine Choppy, editors, *Recent Trends in Data Type Specification. Selected Papers from the 8th Workshop on Specification of Abstract Data Types joint with the 3rd* COMPASS *Workshop*, Dourdan, *Lecture Notes in Computer Science*, volume 655, pages 66–91. Springer, 1993.

Mos96a. Till Mossakowski. Different types of arrow between logical frameworks. In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *Proceedings of the 23rd International Colloquium Automata, Languages and Programming*, Paderborn, *Lecture Notes in Computer Science*, volume 1099, pages 158–169. Springer, 1996.

Mos96b. Till Mossakowski. *Representations, Hierarchies and Graphs of Institutions*. PhD thesis, Universität Bremen, 1996.

Mos00. Till Mossakowski. Specification in an arbitrary institution with symbols. In Didier Bert, Christine Choppy, and Peter D. Mosses, editors, *Recent Trends in Algebraic Development Techniques. Selected Papers from the 14th International Workshop on Algebraic Development Techniques*, Château de Bonas, *Lecture Notes in Computer Science*, volume 1827, pages 252–270. Springer, 2000.

Mos02. Till Mossakowski. Comorphism-based Grothendieck logics. In Krzysztof Diks and Wojciech Rytter, editors, *Proceedings of the 27th Symposium on Mathematical Foundations of Computer Science*, Warsaw, *Lecture Notes in Computer Science*, volume 2420, pages 593–604. Springer, 2002.

Mos03. Till Mossakowski. Foundations of heterogeneous specification. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *Recent Trends in Algebraic Development Techniques.. Selected Papers from the 16th International Workshop on Algebraic Development Techniques*, Frauenchiemsee, *Lecture Notes in Computer Science*, volume 2755, pages 359–375. Springer, 2003.

Mos04. Peter D. Mosses, editor. CASL *Reference Manual*. Number 2960 in Lecture Notes in Computer Science. Springer, 2004.

Mos05. Till Mossakowski. Heterogeneous Specification and the Heterogeneous Tool Set. Habilitation thesis, Universität Bremen, 2005.

MS85. David MacQueen and Donald Sannella. Completeness of proof systems for equational specifications. *IEEE Transactions on Software Engineering*, SE-11(5):454–461, 1985.

MSRR06. Till Mossakowski, Lutz Schröder, Markus Roggenbach, and Horst Reichel. Algebraic-coalgebraic specification in COCASL. *Journal of Logic and Algebraic Programming*, 67(1–2):146–197, 2006.

MSS90. Vincenzo Manca, Antonino Salibra, and Giuseppe Scollo. Equational type logic. *Theoretical Computer Science*, 77(1–2):131–159, 1990.

MST04. Till Mossakowski, Donald Sannella, and Andrzej Tarlecki. A simple refinement language for CASL. In José Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques.. Selected Papers from the 17th International Workshop on Algebraic Development Techniques*, Barcelona, *Lecture Notes in Computer Science*, volume 3423, pages 162–185. Springer, 2004.

MT92. Karl Meinke and John Tucker. Universal algebra. In Samson Abramsky, Dov Gabbay, and Tom Maibaum, editors, *Handbook of Logic in Computer Science. Volume 1 (Background: Mathematical Structures)*, pages 189–409. Oxford University Press, 1992.

MT93. V. Wiktor Marek and Mirosław Truszczyński. *Nonmonotonic Logics: Context-Dependent Reasoning*. Springer, 1993.

MT94. David B. MacQueen and Mads Tofte. A semantics for higher-order functors. In Donald Sannella, editor, *Proceedings of the 5th European Symposium on Programming*, Edinburgh, *Lecture Notes in Computer Science*, volume 788, pages 409–423. Springer, 1994.

MT09. Till Mossakowski and Andrzej Tarlecki. Heterogeneous logical environments for distributed specifications. In Andrea Corradini and Ugo Montanari, editors, *Recent Trends in Algebraic Development Techniques.. Selected Papers from the 19th International Workshop on Algebraic Development Techniques*, Pisa, *Lecture Notes in Computer Science*, volume 5486, pages 266–289. Springer, 2009.

MTD09. Till Mossakowski, Andrzej Tarlecki, and Răzvan Diaconescu. What is a logic translation? *Logica Universalis*, 3(1):95–124, 2009.

MTHM97. Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

MTP97. Till Mossakowski, Andrzej Tarlecki, and Wiesław Pawłowski. Combining and representing logical systems. In Eugenio Moggi and Giuseppe Rosolini, editors, *Proceedings of the 7th International Conference on Category Theory and Computer Science*,

Santa Margherita Ligure, *Lecture Notes in Computer Science*, volume 1290, pages 177–196. Springer, 1997.

MTP98. Till Mossakowski, Andrzej Tarlecki, and Wiesław Pawłowski. Combining and representing logical systems using model-theoretic parchments. In Francesco Parisi-Presicce, editor, *Recent Trends in Data Type Specification. Selected Papers from the 12th International Workshop on Specification of Abstract Data Types*, Tarquinia, *Lecture Notes in Computer Science*, volume 1376, pages 349–364. Springer, 1998.

MTW88. Bernhard Möller, Andrzej Tarlecki, and Martin Wirsing. Algebraic specifications of reachable higher-order algebras. In Donald Sannella and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification. Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, *Lecture Notes in Computer Science*, volume 332, pages 154–169. Springer, 1988.

Mus80. David Musser. On proving inductive properties of abstract data types. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, Las Vegas, pages 154–162, 1980.

MW98. Alfio Martini and Uwe Wolter. A single perspective on arrows between institutions. In Armando Haeberer, editor, *Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology*, Manaus, *Lecture Notes in Computer Science*, volume 1548, pages 486–501. Springer, 1998.

Nel91. Greg Nelson, editor. *Systems Programming in Modula-3*. Prentice-Hall, 1991.

Nip86. Tobias Nipkow. Non-deterministic data types: Models and implementations. *Acta Informatica*, 22(6):629–661, 1986.

NO88. Pilar Nivela and Fernando Orejas. Initial behaviour semantics for algebraic specifications. In Donald Sannella and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification. Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, *Lecture Notes in Computer Science*, volume 332, pages 184–207. Springer, 1988.

Nou81. Farshid Nourani. On induction for programming logic: Syntax, semantics, and inductive closure. *Bulletin of the European Association for Theoretical Computer Science*, 13:51–64, 1981.

Oka98. Chris Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.

ONS93. Fernando Orejas, Marisa Navarro, and Ana Sánchez. Implementation and behavioural equivalence: A survey. In Michel Bidoit and Christine Choppy, editors, *Recent Trends in Data Type Specification. Selected Papers from the 8th Workshop on Specification of Abstract Data Types joint with the 3rd* COMPASS *Workshop*, Dourdan, *Lecture Notes in Computer Science*, volume 655, pages 93–125. Springer, 1993.

Ore83. Fernando Orejas. Characterizing composability of abstract implementations. In Marek Karpinski, editor, *Proceedings of the 1983 International Conference on Foundations of Computation Theory*, Borgholm, *Lecture Notes in Computer Science*, volume 158, pages 335–346. Springer, 1983.

Pad85. Peter Padawitz. Parameter preserving data type specifications. In Hartmut Ehrig, Christiane Floyd, Maurice Nivat, and James Thatcher, editors, *TAPSOFT'85: Proceedings of the International Joint Conference on Theory and Practice of Software Development. Volume 2: Colloquium on Software Engineering*, Berlin, *Lecture Notes in Computer Science*, volume 186, pages 323–341. Springer, 1985.

Pad99. Peter Padawitz. Proof in flat specifications. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 10, pages 321–384. Springer, 1999.

Pau87. Laurence Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, 1987.

Pau96. Laurence Paulson. *ML for the Working Programmer*. Cambridge University Press, second edition, 1996.

Paw96. Wiesław Pawłowski. Context institutions. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification. Selected Papers from*

the 11th Workshop on Specification of Abstract Data Types, Oslo, *Lecture Notes in Computer Science*, volume 1130, pages 436–457. Springer, 1996.

Pet10. Marius Petria. *Generic Refinements for Behavioural Specifications*. PhD thesis, University of Edinburgh, School of Informatics, 2010.

Pey03. Simon Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

Pho92. Wesley Phoa. An introduction to fibrations, topos theory, the effective topos and modest sets. Technical Report ECS-LFCS-92-208, LFCS, Department of Computer Science, University of Edinburgh, 1992.

Pie91. Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, 1991.

Plo77. Gordon D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977.

Poi86. Axel Poigné. On specifications, theories, and models with higher types. *Information and Control*, 68(1–3):1–46, 1986.

Poi88. Axel Poigné. Foundations are rich institutions, but institutions are poor foundations. In Hartmut Ehrig, Horst Herrlich, Hans-Jörg Kreowski, and Gerhard Preuß, editors, *Proceedings of the International Workshop on Categorical Methods in Computer Science with Aspects from Topology*, Berlin, *Lecture Notes in Computer Science*, volume 393, pages 82–101. Springer, 1988.

Poi90. Axel Poigné. Parametrization for order-sorted algebraic specification. *Journal of Computer and System Sciences*, 40:229–268, 1990.

Poi92. Axel Poigné. Basic category theory. In Samson Abramsky, Dov Gabbay, and Tom Maibaum, editors, *Handbook of Logic in Computer Science. Volume 1 (Background: Mathematical Structures)*, pages 413–640. Oxford University Press, 1992.

Pos47. Emil Post. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 12:1–11, 1947.

PS83. Helmuth Partsch and Ralf Steinbrüggen. Program transformation systems. *ACM Computing Surveys*, 15(3):199–236, 1983.

PŞR09. Andrei Popescu, Traian Florin Şerbănuţă, and Grigore Roşu. A semantic approach to interpolation. *Theoretical Computer Science*, 410(12–13):1109–1128, 2009.

QG93. Xiaolei Qian and Allen Goldberg. Referential opacity in nondeterministic data refinement. *ACM Letters on Programming Languages and Systems*, 2(1–4):233–241, 1993.

Qia93. Zhenyu Qian. An algebraic semantics of higher-order types with subtypes. *Acta Informatica*, 30(6):569–607, 1993.

RAC99. Gianna Reggio, Egidio Astesiano, and Christine Choppy. CASL-LTL: a CASL extension for dynamic systems — summary. Technical Report DISI-TR-99-34, DISI, Università di Genova, 1999.

RB88. David Rydeheard and Rod Burstall. *Computational Category Theory*. Prentice Hall International Series in Computer Science. Prentice Hall, 1988.

Rei80. Horst Reichel. Initially-restricting algebraic theories. In Piotr Dembiński, editor, *Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science*, *Lecture Notes in Computer Science*, volume 88, pages 504–514, Rydzyna, 1980. Springer.

Rei81. Horst Reichel. Behavioural equivalence — a unifying concept for initial and final specification methods. In *Proceedings of the 3rd Hungarian Computer Science Conference*, pages 27–39, 1981.

Rei85. Horst Reichel. Behavioural validity of equations in abstract data types. In *Proceedings of the Vienna Conference on Contributions to General Algebra*, pages 301–324. Teubner-Verlag, 1985.

Rei87. Horst Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras*. Oxford University Press, 1987.

RG98. Grigore Roşu and Joseph A. Goguen. Hidden congruent deduction. In Ricardo Caferra and Gernot Salzer, editors, *Proceedings of the 1998 Workshop on First-Order Theorem Proving*, Vienna, *Lecture Notes in Artificial Intelligence*, volume 1761, pages 251–266. Springer, 1998.

RG00. Grigore Roşu and Joseph A. Goguen. On equational Craig interpolation. *Journal of Universal Computer Science*, 6(1):194–200, 2000.

Rod91. Pieter Hendrik Rodenburg. A simple algebraic proof of the equational interpolation theorem. *Algebra Universalis*, 28:48–51, 1991.

Rog06. Markus Roggenbach. CSP-CASL — a new integration of process algebra and algebraic specification. *Theoretical Computer Science*, 354(1):42–71, 2006.

Roş94. Grigore Roşu. The institution of order-sorted equational logic. *Bulletin of the European Association for Theoretical Computer Science*, 53:250–255, 1994.

Roş00. Grigore Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.

RRS00. Sergei Romanenko, Claudio Russo, and Peter Sestoft. Moscow ML owner's manual. Technical report, Royal Veterinary and Agricultural University, Copenhagen, 2000. Available from `http://www.itu.dk/people/sestoft/mosml/manual.pdf`.

RS63. Helena Rasiowa and Roman Sikorski. *The Mathematics of Metamathematics*. Number 41 in Monografie Matematyczne. Polish Scientific Publishers, 1963.

Rus98. Claudio Russo. *Types for Modules*. PhD thesis, University of Edinburgh, Department of Computer Science, 1998. Also in: *Electronic Notes in Theoretical Computer Science*, 60, 2003.

Rut00. Jan J.M.M. Rutten. Universal coalgebra: A theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

San82. Donald Sannella. *Semantics, Implementation and Pragmatics of Clear, a Program Specification Language*. PhD thesis, University of Edinburgh, Department of Computer Science, 1982.

SB83. Donald Sannella and Rod Burstall. Structured theories in LCF. In Giorgio Ausiello and Marco Protasi, editors, *Proceedings of the 8th Colloquium on Trees in Algebra and Programming*, L'Aquila, *Lecture Notes in Computer Science*, volume 159, pages 377–391. Springer, 1983.

Sch86. David Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, 1986.

Sch87. Oliver Schoett. *Data Abstraction and the Correctness of Modular Programs*. PhD thesis, University of Edinburgh, Department of Computer Science, 1987.

Sch90. Oliver Schoett. Behavioural correctness of data representations. *Science of Computer Programming*, 14(1):43–57, 1990.

Sch92. Oliver Schoett. Two impossibility theorems on behaviour specification of abstract data types. *Acta Informatica*, 29(6/7):595–621, 1992.

Sco76. Dana Scott. Data types as lattices. *SIAM Journal of Computing*, 5(3):522–587, 1976.

Sco04. Giuseppe Scollo. An institution isomorphism for planar graph colouring. In Rudolf Berghammer, Bernhard Möller, and Georg Struth, editors, *Relational and Kleene-Algebraic Methods in Computer Science. Selected Papers from the 7th International Seminar on Relational Methods in Computer Science and 2nd International Workshop on Applications of Kleene Algebra*, Bad Malente, *Lecture Notes in Computer Science*, volume 3051, pages 252–264. Springer, 2004.

SCS94. Amílcar Sernadas, José Félix Costa, and Cristina Sernadas. An institution of object behaviour. In Hartmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification. Selected Papers from the 9th Workshop on Specification of Abstract Data Types joint with the 4th* COMPASS *Workshop*, Caldes de Malavella, *Lecture Notes in Computer Science*, volume 785, pages 337–350. Springer, 1994.

Sel72. Alan Selman. Completeness of calculi for axiomatically defined classes of algebras. *Algebra Universalis*, 2:20–32, 1972.

SH00. Christopher A. Stone and Robert Harper. Deciding type equivalence in a language with singleton kinds. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, Boston, pages 214–227, 2000.

Sha08. Stewart Shapiro. Classical logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. CSLI, Stanford University, fall 2008 edition, 2008. Available from `http://plato.stanford.edu/archives/fall2008/entries/logic-classical/`.

SM09. Lutz Schröder and Till Mossakowski. HASCASL: Integrated higher-order specification and program development. *Theoretical Computer Science*, 410(12–13):1217–1260, 2009.

Smi93. Douglas R. Smith. Constructing specification morphisms. *Journal of Symbolic Computation*, 15(5/6):571–606, 1993.

Smi06. Douglas R. Smith. Composition by colimit and formal software development. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 4060, pages 317–332. Springer, 2006.

SML05. Lutz Schröder, Till Mossakowski, and Christoph Lüth. Type class polymorphism in an institutional framework. In José Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques.. Selected Papers from the 17th International Workshop on Algebraic Development Techniques*, Barcelona, *Lecture Notes in Computer Science*, volume 3423, pages 234–248. Springer, 2005.

Smo86. Gert Smolka. Order-sorted Horn logic: Semantics and deduction. Technical Report SR-86-17, Universität Kaiserslautern, Fachbereich Informatik, 1986.

SMT+05. Lutz Schröder, Till Mossakowski, Andrzej Tarlecki, Bartek Klin, and Piotr Hoffman. Amalgamation in the semantics of CASL. *Theoretical Computer Science*, 331(1):215–247, 2005.

Spi92. J. Michael Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, second edition, 1992.

SS93. Antonino Salibra and Guiseppe Scollo. A soft stairway to institutions. In Michel Bidoit and Christine Choppy, editors, *Recent Trends in Data Type Specification. Selected Papers from the 8th Workshop on Specification of Abstract Data Types joint with the 3rd* COMPASS *Workshop*, Dourdan, *Lecture Notes in Computer Science*, volume 655, pages 310–329. Springer, 1993.

SS96. Antonino Salibra and Giuseppe Scollo. Interpolation and compactness in categories of pre-institutions. *Mathematical Structures in Computer Science*, 6(3):261–286, 1996.

SST92. Donald Sannella, Stefan Sokołowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Parameterisation revisited. *Acta Informatica*, 29(8):689–736, 1992.

ST85. Donald Sannella and Andrzej Tarlecki. Program specification and development in Standard ML. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, New Orleans, pages 67–77, 1985.

ST86. Donald Sannella and Andrzej Tarlecki. Extended ML: An institution-independent framework for formal program development. In David H. Pitt, Samson Abramsky, Axel Poigné, and David E. Rydeheard, editors, *Proceedings of the Tutorial and Workshop on Category Theory and Computer Programming*, Guildford, *Lecture Notes in Computer Science*, volume 240, pages 364–389. Springer, 1986.

ST87. Donald Sannella and Andrzej Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Sciences*, 34:150–178, 1987.

ST88a. Donald Sannella and Andrzej Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76(2/3):165–210, 1988.

ST88b. Donald Sannella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica*, 25:233–281, 1988.

ST89. Donald Sannella and Andrzej Tarlecki. Toward formal development of ML programs: Foundations and methodology. In Josep Díaz and Fernando Orejas, editors, *TAPSOFT'89: Proceedings of the International Joint Conference on Theory and Practice of Software Development. Volume 2: Advanced Seminar on Foundations of Innovative Software Development II and Colloquium on Current Issues in Programming Languages*, Barcelona, *Lecture Notes in Computer Science*, volume 352, pages 375–389. Springer, 1989.

ST97. Donald Sannella and Andrzej Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, 9:229–269, 1997.

ST04. Donald Sannella and Andrzej Tarlecki, editors. CASL semantics. In *[Mos04]*. 2004.

ST06. Donald Sannella and Andrzej Tarlecki. Horizontal composability revisited. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning and Computation: Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science*, volume 4060, pages 296–316. Springer, 2006.

ST08. Donald Sannella and Andrzej Tarlecki. Observability concepts in abstract data type specification, 30 years later. In Pierpaolo Degano, Rocco de Nicola, and José Meseguer, editors, *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of his 65th Birthday*, Lecture Notes in Computer Science. Springer, 2008.

Str67. Christopher Strachey. Fundamental concepts in programming languages. In *NATO Summer School in Programming, Copenhagen*. 1967. Also in: *Higher-Order and Symbolic Computation* 13(1–2):11–49, 2000.

SU06. Morten H. Sørensen and Paweł Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Number 149 in Studies in Logic and the Foundations of Mathematics. Elsevier Science, 2006.

SW82. Donald Sannella and Martin Wirsing. Implementation of parameterised specifications. In Mogens Nielsen and Erik Meineche Schmidt, editors, *Proceeding of the 9th International Colloquium on Automata, Languages and Programming*, Aarhus, *Lecture Notes in Computer Science*, volume 140, pages 473–488. Springer, 1982.

SW83. Donald Sannella and Martin Wirsing. A kernel language for algebraic specification and implementation. In Marek Karpinski, editor, *Proceedings of the 1983 International Conference on Foundations of Computation Theory*, Borgholm, *Lecture Notes in Computer Science*, volume 158, pages 413–427. Springer, 1983.

SW99. Donald Sannella and Martin Wirsing. Specification languages. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 8, pages 243–272. Springer, 1999.

Tar85. Andrzej Tarlecki. On the existence of free models in abstract algebraic institutions. *Theoretical Computer Science*, 37(3):269–304, 1985.

Tar86a. Andrzej Tarlecki. Bits and pieces of the theory of institutions. In David H. Pitt, Samson Abramsky, Axel Poigné, and David E. Rydeheard, editors, *Proceedings of the Tutorial and Workshop on Category Theory and Computer Programming*, Guildford, *Lecture Notes in Computer Science*, volume 240, pages 334–360. Springer, 1986.

Tar86b. Andrzej Tarlecki. Quasi-varieties in abstract algebraic institutions. *Journal of Computer and System Sciences*, 33(3):333–360, 1986.

Tar87. Andrzej Tarlecki. Institution representation. Unpublished note, Dept. of Computer Science, University of Edinburgh, 1987.

Tar96. Andrzej Tarlecki. Moving between logical systems. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification. Selected Papers from the 11th Workshop on Specification of Abstract Data Types*, Oslo, *Lecture Notes in Computer Science*, volume 1130, pages 478–502. Springer, 1996.

Tar99. Andrzej Tarlecki. Institutions: An abstract framework for formal specification. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 4, pages 105–130. Springer, 1999.

Tar00. Andrzej Tarlecki. Towards heterogeneous specifications. In Dov Gabbay and Maarten de Rijke, editors, *Frontiers of Combining Systems 2*, Studies in Logic and Computation, pages 337–360. Research Studies Press, 2000.

TBG91. Andrzej Tarlecki, Rod M. Burstall, and Joseph A. Goguen. Some fundamental algebraic tools for the semantics of computation. Part 3: Indexed categories. *Theoretical Computer Science*, 91(2):239–264, 1991.

Ter03. Terese. *Term Rewriting Systems*, *Cambridge Tracts in Theoretical Computer Science*, volume 55. Cambridge University Press, 2003.

Tho89. Simon Thompson. A logic for Miranda. *Formal Aspects of Computing*, 1(4):339–365, 1989.

TM87. Władysław M. Turski and Thomas S.E. Maibaum. *Specification of Computer Programs*. Addison-Wesley, 1987.

Tra93. Will Tracz. Parametrized programming in LILEANNA. In *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, Indianapolis, pages 77–86, 1993.

TWW82. James Thatcher, Eric Wagner, and Jesse Wright. Data type specification: Parameterization and the power of specification techniques. *ACM Transactions on Programming Languages and Systems*, 4(4):711–732, 1982.

Vra88. Jos L.M. Vrancken. The algebraic specification of semi-computable data types. In Donald Sannella and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification. Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, *Lecture Notes in Computer Science*, volume 332, pages 249–259. Springer, 1988.

Wad89. Philip Wadler. Theorems for free! In *Proceedings of the 4th International ACM Conference on Functional Programming Languages and Computer Architecture*, London, pages 347–359, 1989.

Wan79. Mitchell Wand. Final algebra semantics and data type extensions. *Journal of Computer and System Sciences*, 19:27–44, 1979.

Wan82. Mitchell Wand. Specifications, models, and implementations of data abstractions. *Theoretical Computer Science*, 20(1):3–32, 1982.

WB82. Martin Wirsing and Manfred Broy. An analysis of semantic models for algebraic specifications. In Manfred Broy and Gunther Schmidt, editors, *Theoretical Foundations of Programming Methodology: Lecture Notes of an International Summer School, Marktoberdorf 1981*, pages 351–416. Reidel, 1982.

WB89. Martin Wirsing and Manfred Broy. A modular framework for specification and implementation. In Josep Díaz and Fernando Orejas, editors, *TAPSOFT'89: Proceedings of the International Joint Conference on Theory and Practice of Software Development. Volume 1: Advanced Seminar on Foundations of Innovative Software Development I and Colloquium on Trees in Algebra and Programming*, Barcelona, *Lecture Notes in Computer Science*, volume 351, pages 42–73. Springer, 1989.

WE87. Eric G. Wagner and Hartmut Ehrig. Canonical constraints for parameterized data types. *Theoretical Computer Science*, 50:323–349, 1987.

Wec92. Wolfgang Wechler. *Universal Algebra for Computer Scientists*, *EATCS Monographs on Theoretical Computer Science*, volume 25. Springer, 1992.

Wik. Wikipedia. Hash table. Available from `http://en.wikipedia.org/wiki/Hash_table`.

Wir82. Martin Wirsing. Structured algebraic specifications. In *Proceedings of the AFCET Symposium on Mathematics for Computer Science*, Paris, pages 93–107, 1982.

Wir86. Martin Wirsing. Structured algebraic specifications: A kernel language. *Theoretical Computer Science*, 42(2):123–249, 1986.

Wir90. Martin Wirsing. Algebraic specification. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B (Formal Models and Semantics)*, pages 675–788. North-Holland and MIT Press, 1990.

Wir93. Martin Wirsing. Structured specifications: Syntax, semantics and proof calculus. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and*

Algebra of Specification: Proceedings of the NATO Advanced Study Institute, Marktoberdorf 1991, pages 411–442. Springer, 1993.

WM97. Michal Walicki and Sigurd Meldal. Algebraic approches to nondeterminism: An overview. *ACM Computing Surveys*, 29(1):30–81, 1997.

Zil74. Steven Zilles. Abstract specification of data types. Technical Report 119, Computation Structures Group, Massachusetts Institute of Technology, 1974.

Zuc99. Elena Zucca. From static to dynamic abstract data-types: An institution transformation. *Theoretical Computer Science*, 216(1–2):109–157, 1999.