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) $i1{:}X \to X \uplus Y$ and $i2{:}Y \to X \uplus Y$ 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 $(\prod \langle X_i \rangle_{i \in I})_s = \{ f{:}I \to \bigcup_{i \in I} (X_i)_s \mid f(i) \in (X_i)_s \text{ for all } i \in I\}$ 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* $f{:}X \to Y$ is an S-indexed family of functions $f = \langle f_s{:}X_s \to Y_s \rangle_{s \in S}$; X is called the *domain* (or *source*) of f, and Y is called its *codomain* (or *target*). An S-sorted function $f{:}X \to Y$ is an *identity* (an *inclusion*, *surjective*, *injective*, *bijective*, ...) if for every $s \in S$, the function $f_s{:}X_s \to Y_s$ 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: xR_sx;
- symmetric: xR_sy implies yR_sx; and
- transitive: xR_sy and yR_sz implies xR_sz

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:

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

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.

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

(i.e., $f_{s_1}(\Box) = 1$ 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

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

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

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

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 $f\colon s_1 \times \cdots \times s_n \to s$ 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 $f\colon s$ will be used for $f\colon \varepsilon \to s$ (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 $f\colon s_1 \times \cdots \times s_n \to t_1 \times \cdots \times t_m$ may be viewed as a family of m functions

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

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 $f\colon s_1 \times \cdots \times s_n \to s$ and $f\colon t_1 \times \cdots \times t_m \to t$ 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 $arity\colon \Omega \to S^*$ and $sort\colon \Omega \to S$. 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 $f\colon s_1 \times \cdots \times s_n \to s$ in Σ, a function (or *operation*) $(f\colon s_1 \times \cdots \times s_n \to s)_A\colon |A|_{s_1} \times \cdots \times |A|_{s_n} \to |A|_s$. □

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 $f\colon s_1 \times \cdots \times s_n \to s$ is in Σ then the operation $(f\colon s_1 \times \cdots \times s_n \to s)_A$ is a function on the corresponding carrier sets of A. If $n = 0$ (i.e. $f\colon s$), then $|A|_{s_1} \times \cdots \times |A|_{s_n}$ is a singleton set containing the empty tuple $\langle\rangle$, and then $(f\colon s)_A$ may be viewed as a constant denoting the value $(f\colon s)_A(\langle\rangle) \in |A|_s$. Notice that $(f\colon s_1 \times \cdots \times s_n \to s)_A$ 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 $f\colon s_1 \times \cdots \times s_n \to s$ be in Σ. We always write f_A in place of $(f\colon s_1 \times \cdots \times s_n \to s)_A$ when there is no danger of confusion. When $n = 0$ (i.e. $f\colon s$), we write $(f\colon s)_A$ or f_A in place of $(f\colon s)_A(\langle\rangle)$. □

[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 $S1 = \{shape, suit\}$ and let $\Omega 1_{\varepsilon,shape} = \{box\}$, $\Omega 1_{\varepsilon,suit} = \{hearts\}$, $\Omega 1_{shape,shape} = \{boxify\}$, $\Omega 1_{shape\,suit,suit} = \{f\}$, 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 $box{:}shape$, $hearts{:}suit$, $boxify{:}shape \to shape$ and $f{:}shape \times suit \to suit$. We can present $\Sigma 1$ in tabular form as follows (this notation will be used later with the obvious meaning):

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

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

$$\begin{array}{l}|A1|_{shape} = \{\Box, \triangle\}, \\ |A1|_{suit} = \{\clubsuit, \heartsuit, \spadesuit\}, \\ box_{A1} = \Box \in |A1|_{shape}, \\ hearts_{A1} = \heartsuit \in |A1|_{suit}, \\ boxify_{A1}{:}|A1|_{shape} \to |A1|_{shape} = \{\Box \mapsto \Box, \triangle \mapsto \Box\},\end{array}$$

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

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

(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 $f{:}s_1 \times \cdots \times s_n \to s$ 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 $f{:}s_1 \times \cdots \times s_n \to s$ 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

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

$B1$ 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 $f \colon s_1 \times \cdots \times s_n \to s$ 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 $f \colon s_1 \times \cdots \times s_n \to s$ 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

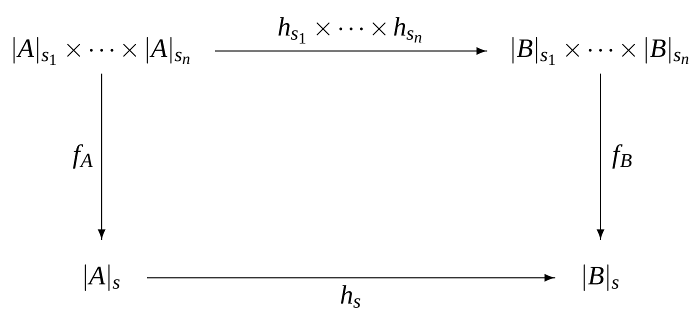
A homomorphism between algebras is the analogue of a function between sets, and a congruence relation on an algebra is the analogue of an equivalence relation on a set. An algebra has more structure than a set, so homomorphisms and congruences are required to respect the additional structure (i.e. the 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* $h{:}A \to B$ is an S-sorted function $h{:}|A| \to |B|$ which respects the operations of Σ, i.e. such that for all $f{:}s_1 \times \cdots \times s_n \to s$ in Σ and $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$, $h_s(f_A(a_1, \ldots, a_n)) = f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$. A Σ-homomorphism $h{:}A \to B$ is an *identity* (an *inclusion*, *surjective*, ...) if it is an identity (an inclusion, surjective, ...) when viewed as an S-sorted function. □

Notation. If $h{:}A \to B$ is a Σ-homomorphism, then $|h|{:}|A| \to |B|$ 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:



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

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

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

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

Let $h1\colon |A1| \to |C1|$ 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 $h1\colon A1 \to C1$ 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 $id_{|A|}\colon A \to A$ (the identity S-sorted function) is a Σ-homomorphism. Let $h\colon A \to B$ and $h'\colon B \to C$ be Σ-homomorphisms. Show that $|h|;|h'|\colon |A| \to |C|$ is a Σ-homomorphism $h;h'\colon A \to C$. □

Exercise 1.3.4. Let $h\colon A \to B$ 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 $f\colon s_1 \times \cdots \times s_n \to s$ in Σ and $a_1 \in |A'|_{s_1}, \ldots, a_n \in |A'|_{s_n}$, $f_{h(A')}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) = h_s(f_{A'}(a_1, \ldots, a_n))$.

Show that $h(A')$ is a well-defined Σ-algebra (in particular, that the function $f_{h(A')}\colon |h(A')|_{s_1} \times \cdots \times |h(A')|_{s_n} \to |h(A')|_s$ is well-defined for each $f\colon s_1 \times \cdots \times s_n \to s$ 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 $h\colon A \to B$ 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 $h\colon A \to B$ 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 $h{:}B \to A$, and that any Σ-homomorphism $h{:}A \to B$ is surjective. □

Definition 1.3.7 (Isomorphism). Let A and B be Σ-algebras. A Σ-homomorphism $h{:}A \to B$ is a Σ-*isomorphism* if it has an inverse, i.e. there is a Σ-homomorphism $h^{-1}{:}B \to A$ 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 $h{:}A \cong B$ or just $A \cong B$. □

Exercise 1.3.8. Let $h{:}A \cong B$ and $h'{:}B \cong C$ be Σ-isomorphisms. Show that their composition is a Σ-isomorphism $h;h'{:}A \cong C$. 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

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

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

f_{D1}	1	2	3
$\Box$	2	3	3
$\triangle$	1	3	2

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

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

This defines a $\Sigma 1$-homomorphism $i1{:}A1 \to D1$ 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

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

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 $\ldots$ 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 $f\colon s_1 \times \cdots \times s_n \to s$ 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

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

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

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

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

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

□

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 $\langle h_s\colon |A|_s \to (|A|/{\equiv})_s\rangle_{s \in S}$ is a Σ-homomorphism $h\colon A \to A/{\equiv}$ with $\ker(h) = {\equiv}$. □

Exercise 1.3.19. Let $h\colon A \to B$ 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 $h\colon A \to B$ such that $\equiv \subseteq \ker(h)$, there exists a unique Σ-homomorphism $g\colon A/{\equiv} \to B$ 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 $h\colon A \to B$ 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 $h\colon A \to B$ be a Σ-homomorphism. Show that there is a unique Σ-congruence $\equiv$ on A and a unique injective Σ-homomorphism $g\colon A/{\equiv} \to B$ 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

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

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

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

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

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

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

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

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 $\Sigma 2 = \langle S2, \Omega 2\rangle$

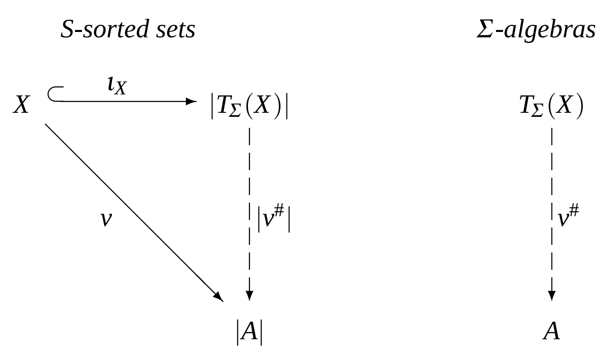
be the signature with sorts s, s', t and operations $a{:}\,s$, $a{:}\,s'$, $f{:}\,s \to t$ and $f{:}\,s' \to t$ (no mistakes here, repetition of names is intented).

According to the definition, $|T_{\Sigma 2}|_t = \{$“$f(a():s):t$”, “$f(a():s'):t$”$\}$. 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 $\iota;v{:}X \to |A|$). $\square$

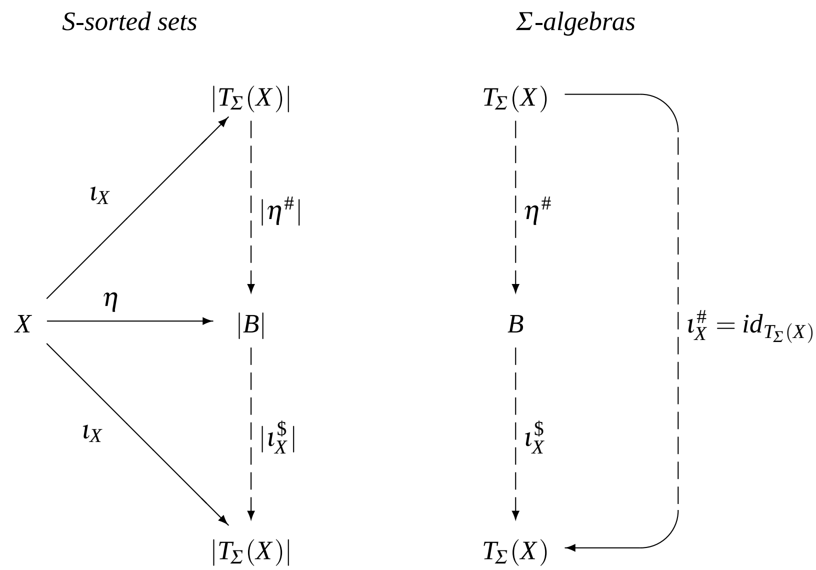
Exercise 1.4.7. Define evaluation of terms in an inductive fashion. Convince yourself that the result is the same as that given by Definition 1.4.5. $\square$

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. $\square$

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'. $\square$

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. $\square$

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 $X \to Y_1 \dots Y_n$ in G corresponds to an operation in Ω_G with result sort X and arity given by the sequence of non-terminal symbols in $Y_1 \dots Y_n$. The Σ_G-algebra A_G has carriers $|A_G|_X = T^*$ for all $X \in S_G$, and for any $p{:}X_1 \times \dots \times X_n \to X$ in Σ_G and $a_1, \dots, a_n \in T^*$, $p_{A_G}(a_1, \dots, a_n)$ 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 $\sigma_{ops} = \langle \sigma_{w,s}\colon \Omega_{w,s} \to \Omega'_{\sigma^*_{sorts}(w),\sigma_{sorts}(s)} \rangle_{w\in S^*, s\in S}$ (where for $w = s_1 \dots s_n \in S^*$, $\sigma^*_{sorts}(w) = \sigma_{sorts}(s_1)\dots\sigma_{sorts}(s_n)$). A signature morphism $\sigma\colon \Sigma \to \Sigma'$ is a *signature inclusion* $\sigma\colon \Sigma \hookrightarrow \Sigma'$ 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 $\sigma\colon \Sigma \to \Sigma'$, 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.

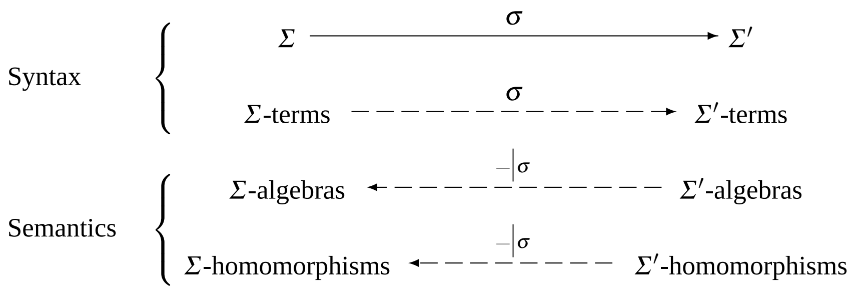
Define $\sigma_{sorts}\colon S \to S1$ and $\sigma_{ops} = \langle \sigma_{w,s}\colon \Omega_{w,s} \to \Omega 1_{\sigma^*_{sorts}(w),\sigma_{sorts}(s)} \rangle_{w\in S^*, s\in S}$ by

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

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

Exercise 1.5.3. Let $\sigma\colon \Sigma \to \Sigma'$ and $\sigma'\colon \Sigma' \to \Sigma''$ 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 $\sigma;\sigma'\colon \Sigma \to \Sigma''$. □

In the rest of this section, let $\sigma\colon \Sigma \to \Sigma'$ 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

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

□

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

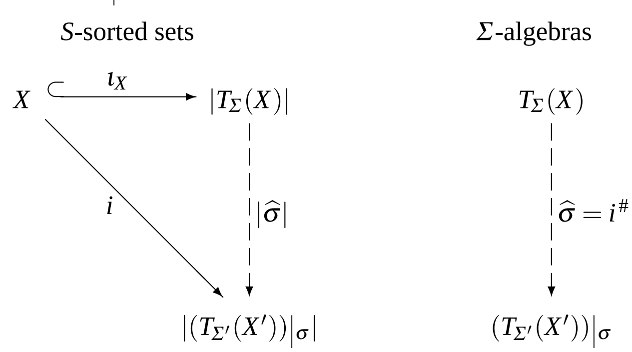
Exercise 1.5.9. Define the σ-reduct $\equiv'|_\sigma$ of a Σ'-congruence $\equiv'$ on a Σ'-algebra A', and prove that it is a Σ-congruence on $A'|_\sigma$. Show that σ-reduct distributes over quotient, i.e. $(A'/{\equiv'})|_\sigma = (A'|_\sigma)/(\equiv'|_\sigma)$ for all Σ'-algebras A' and Σ'-congruences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

The following definition of the translation of terms along a signature morphism $\sigma\colon \Sigma \to \Sigma'$ 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 $i\colon X \to |(T_{\Sigma'}(X'))|_\sigma|$ 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 $\widehat{\sigma}\colon T_\Sigma(X) \to (T_{\Sigma'}(X'))|_\sigma$ 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 $\sigma\colon \Sigma \to \Sigma 1$ 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 $x\colon \mathit{polygon}, x\colon \mathit{figure}, y\colon \mathit{figure}, z\colon \mathit{trump}$. The $S1$-sorted set of variables X' in Definition 1.5.10 is then $x\colon \mathit{shape}, x'\colon \mathit{shape}, y\colon \mathit{shape}, z\colon \mathit{suit}$, 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 $s_1 \dots s_n \in S^*$, let $I_{s_1 \dots s_n}$ be the S-sorted set $\boxed{1}{:}s_1, \dots, \boxed{n}{:}s_n$. The *derived signature of* Σ is the signature $\Sigma^{der} = \langle S, \Omega^{der} \rangle$ where for each $s_1 \dots s_n \in S^*$ and $s \in S$, $\Omega^{der}_{s_1 \dots s_n, s} = |T_\Sigma(I_{s_1 \dots s_n})|_s$. □

In the derived signature of Σ, a Σ-term t of sort s with variables $I_{s_1 \dots s_n}$ represents an operation $t{:}s_1 \times \dots \times s_n \to s$. The variable $\boxed{i}{:}s_i$ in $I_{s_1 \dots s_n}$ thus stands for the ith argument of t. Note that a "bare" variable $\boxed{i} \in |T_\Sigma(I_{s_1 \dots s_n})|_{s_i}$ is an operation $i{:}s_1 \times \dots \times s_n \to s_i$ 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 $t{:}s_1 \times \dots \times s_n \to s$ in Σ^{der} and $a_1 \in |A^{der}|_{s_1}, \dots, a_n \in |A^{der}|_{s_n}$, $t_{A^{der}}(a_1, \dots, a_n) = t_A(v) \in |A^{der}|_s$ where v is the S-sorted function $\{(\boxed{1}{:}s_1) \mapsto a_1, \dots, (\boxed{n}{:}s_n) \mapsto a_n\}$. □

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

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

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

$h_{A1\vert_\delta}$	$\clubsuit$	$\heartsuit$	$\spadesuit$
$\Box$	$\clubsuit$	$\heartsuit$	$\spadesuit$
$\triangle$	$\spadesuit$	$\heartsuit$	$\heartsuit$

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 $\delta\colon\Sigma \to \Sigma'$ 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].

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.