Donald Sannella and Andrzej Tarlecki

Foundations of Algebraic Specification and Formal Software Development

September 29, 2010

Springer

Contents

Chapter 2
Simple equational specifications

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

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

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

2.1 Equations

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Exercise 2.1.10. Prove the Satisfaction Lemma, using Exercise 1.5.12. $\square$

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

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

2.2 Flat specifications

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

Throughout this section, let Σ be a signature.

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

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

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

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

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

$$
\begin{array}{lll}
\textbf{spec}\ \mathrm{BOOL} = & \textbf{sorts} & bool \\
& \textbf{ops} & true : bool \\
& & false : bool \\
& & \neg \_\_ : bool \to bool \\
& & \_\_ \wedge \_\_ : bool \times bool \to bool \\
& & \_\_ \Rightarrow \_\_ : bool \times bool \to bool \\
& \multicolumn{2}{l}{\forall p, q : bool} \\
& & \bullet\ \neg true = false \\
& & \bullet\ \neg false = true \\
& & \bullet\ p \wedge true = p \\
& & \bullet\ p \wedge \neg p = false \\
& & \bullet\ p \Rightarrow q = \neg(p \wedge \neg q)
\end{array}
$$

Define $\Sigma\mathrm{BOOL}$-algebras $A1$, $A2$ and $A3$ as follows:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$
\begin{array}{lll}
\textbf{spec } \mathrm{BA} = & \textbf{sorts} & \mathit{bool} \\
& \textbf{ops} & \mathsf{true}\colon \mathit{bool} \\
& & \mathsf{false}\colon \mathit{bool} \\
& & \neg\_\_\colon \mathit{bool} \to \mathit{bool} \\
& & \_\_\vee\_\_\colon \mathit{bool} \times \mathit{bool} \to \mathit{bool} \\
& & \_\_\wedge\_\_\colon \mathit{bool} \times \mathit{bool} \to \mathit{bool} \\
& & \_\_\Rightarrow\_\_\colon \mathit{bool} \times \mathit{bool} \to \mathit{bool} \\
& \multicolumn{2}{l}{\forall p,q,r\colon \mathit{bool}} \\
& & \bullet\ p \vee (q \vee r) = (p \vee q) \vee r \\
& & \bullet\ p \wedge (q \wedge r) = (p \wedge q) \wedge r \\
& & \bullet\ p \vee q = q \vee p \\
& & \bullet\ p \wedge q = q \wedge p \\
& & \bullet\ p \vee (p \wedge q) = p \\
& & \bullet\ p \wedge (p \vee q) = p \\
& & \bullet\ p \vee (q \wedge r) = (p \wedge q) \vee (p \wedge r) \\
& & \bullet\ p \wedge (q \vee r) = (p \vee q) \wedge (p \vee r) \\
& & \bullet\ p \vee \neg p = \mathsf{true} \\
& & \bullet\ p \wedge \neg p = \mathsf{false} \\
& & \bullet\ p \Rightarrow q = \neg p \vee q
\end{array}
$$

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

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

and

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Example 2.2.11. Consider the signature

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

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

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

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

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

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

□

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

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

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

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

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

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

A final remark to clarify the nuances in the many-sorted versions of Theorem 2.2.10 is that the theorem holds for *any* signature (also with an infinite set of sort names) when we restrict attention to algebras with non-empty carriers of all sorts: all varieties of such algebras (with closure under subalgebras limited to subalgebras with non-empty carriers) are definable by equations with a finite set of variables.

2.3 Theories

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

Throughout this section, let Σ be a signature.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The fact that 2.3.13(2) implies 2.3.13(1) gives a shortcut for checking if a signature morphism is a theory morphism: one need only check, for each axiom in some *presentation* of the source theory, that the translation of that axiom is in the target theory. The equivalence between 2.3.13(1) and 2.3.13(3) is similar in spirit to the Satisfaction Lemma, demonstrating a perfect correspondence between translation of syntax (axioms) along a signature morphism and translation of semantics (models) in the opposite direction. This equivalence shows that there is a model-level alternative to the axiom-level phrasing of Definition 2.3.11; in fact, we will take this alternative in the case of structured specifications (Chapter 5) where there is no equivalent axiom-level characterisation (Exercise 5.5.4).

Example 2.3.14. Let Σ be the signature

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

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

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

Let Φ be the set of Σ-equations

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

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

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

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

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

2.4 Equational calculus

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

Throughout this section, let Σ be a signature.

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

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

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

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

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

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

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

□

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

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

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

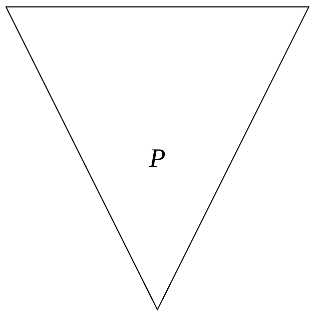
Prove that if $\Psi \vdash_\Sigma \varphi$ and $\{\varphi\} \cup \Phi \vdash_\Sigma \psi$ then $\Psi \cup \Phi \vdash_\Sigma \psi$. (HINT: Use induction on the structure of the derivation of $\{\varphi\} \cup \Phi \vdash_\Sigma \psi$; for the case of the axiom rule, use the fact that weakening is admissible.)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

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

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

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

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



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

where P is the derivation

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Completeness ($\Phi \models_\Sigma \varphi \Rightarrow \Phi \vdash_\Sigma \varphi$) is typically more difficult to achieve than soundness: it means that the rules in the system are powerful enough to derive all correct results. It is not as important as soundness, in the sense that a complete but unsound system is useless while (as we shall see in the sequel) a sound but incomplete system is often the best that can be obtained. The equational calculus happens to be complete for $\models_\Sigma$:

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

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

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

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

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

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

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

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

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

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

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

□

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

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

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

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

2.5 Initial models

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

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

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

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

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

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

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

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

$$
\begin{array}{lll}
\textbf{spec } \textsc{Nat} = & \textbf{sorts} & nat \\
 & \textbf{ops} & 0{:}\, nat \\
 & & succ{:}\, nat \to nat \\
 & & \_\_ + \_\_{:}\, nat \times nat \to nat \\
 & \forall m,n{:}\, nat & \bullet\ 0 + n = n \\
 & & \bullet\ succ(m) + n = succ(m+n)
\end{array}
$$

List some of the models of NAT. Which of these contain junk and/or confusion? (NOTE: For reference later in this section, ΣNAT refers to the signature of NAT and ΦNAT refers to its axioms.) □

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

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

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

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

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

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

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

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

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

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

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

where

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

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

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

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

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

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

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

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

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

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

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

(**Exercise:** Check uniqueness.)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{array}{ll}\textbf{spec } \mathrm{E\scriptstyle XPR} = & \textbf{sorts } \mathit{expr} \\ & \textbf{ops} \quad x, y, 0 \colon \mathit{expr} \\ & \qquad\quad \mathit{plus}, \mathit{minus} \colon \mathit{expr} \times \mathit{expr} \to \mathit{expr} \\ & \forall e, e' \colon \mathit{expr} \bullet \mathit{plus}(e, e') = \mathit{plus}(e', e)\end{array}$$

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

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

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

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

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

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

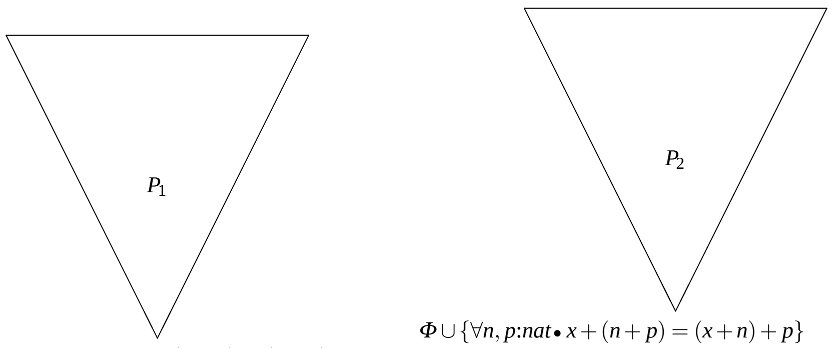
$$\frac{\Phi \vdash_{\Sigma} \forall X \bullet t = t'}{\Phi \vdash_{\Sigma \cup \Sigma'} \forall X \bullet t = t'}$$

Exercise. Show that adding the two inference rules above to the equational calculus gives a system that is sound for reachable models of $\Sigma\mathrm{NAT}$-presentations.

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

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

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



$$\frac{\Phi \vdash_{\Sigma\textsc{Nat}} \forall n,p{:}nat\bullet 0+(n+p)=(0+n)+p \qquad \begin{array}{l}\Phi \cup \{\forall n,p{:}nat\bullet x+(n+p)=(x+n)+p\}\\ \quad\vdash_{\Sigma\textsc{Nat}\cup\{x:nat\}}\\ \qquad \forall n,p{:}nat\bullet succ(x)+(n+p)=(succ(x)+n)+p\end{array}}{\Phi \vdash_{\Sigma\textsc{Nat}} \forall x,n,p{:}nat\bullet x+(n+p)=(x+n)+p}$$

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

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

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

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

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

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

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

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

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

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

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

A generalisation of the concept of initial model is needed to give a fully satisfactory specification of classes of models that are naturally parametric with respect to some basic data. An example is the definition of terms in Section 1.4, which is parametric in an S-sorted set of variables. Another is the specification of sets (see Exercise 2.5.23): it should be possible to specify sets without building in a specification of the kind of values in the sets (in this case, natural numbers).

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

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

2.6 Term rewriting

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{aligned}\mathrm{RBOOL} = \{&\neg true \rightarrow false,\ \neg false \rightarrow true,\ p \wedge true \rightarrow p,\ p \wedge false \rightarrow false,\\ &p \wedge \neg p \rightarrow false,\ p \Rightarrow q \rightarrow \neg(p \wedge \neg q)\}.\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

Proof. **Exercise:** Do it. □

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

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

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

where $f{:}s_1 \times \cdots \times s_n \to s$. By the inductive assumption, $t_1 \sim_R t'_1$ and $\ldots$ and $t_n \sim_R t'_n$. Then, by repeated application of Lemma 2.6.9(1), we have $f(t_1, t_2 \ldots, t_n) \sim_R f(t'_1, t_2 \ldots, t_n) \sim_R \cdots \sim_R f(t'_1, t'_2 \ldots, t'_n)$ (using first the context $f(\Box{:}s_1, t_2 \ldots, t_n)$, then $f(t'_1, \Box{:}s_2, \ldots, t_n)$, then $\ldots$, then $f(t'_1, t'_2, \ldots, \Box{:}s_n)$). When the last step of the derivation of $Eq(R) \vdash_\Sigma t = t'$ is an application of the instantiation rule, the result follows directly by Lemma 2.6.9(2). (**Exercise:** Complete the proof.) $\Box$

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

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

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

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

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

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

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

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

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

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

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

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

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

Exercise 2.6.18. Is your term rewriting system $\mathrm{RN}\textsc{at}$ from Exercise 2.6.7 complete? If not, find an alternative term rewriting system for $\mathrm{N}\textsc{at}$ that is complete. □

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.7 Fiddling with the definitions

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

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

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

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

2.7.1 Conditional equations

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

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

Definition 2.7.1 (Conditional equation). A *(positive) conditional Σ-equation* $\forall X \bullet t_1 = t_1' \wedge \ldots \wedge t_n = t_n' \Rightarrow t_0 = t_0'$ consists of:

- a finite S-sorted set X (of variables), such that $X_s \subseteq \mathscr{X}$ for all $s \in S$; and
- for each $0 \leq j \leq n$ (where $n \geq 0$), two Σ-terms $t_j, t_j' \in |T_\Sigma(X)|_{s_j}$ for some sort $s_j \in S$.

A Σ-algebra A *satisfies* a conditional Σ-equation $\forall X \bullet t_1 = t_1' \wedge \ldots \wedge t_n = t_n' \Rightarrow t_0 = t_0'$ if for every (S-sorted) function $v{:}X \to |A|$, if $(t_1)_A(v) = (t_1')_A(v)$ and $\ldots$ and $(t_n)_A(v) = (t_n')_A(v)$ then $(t_0)_A(v) = (t_0')_A(v)$. □

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

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

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

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

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

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

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

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

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

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

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

- $p \vee (q \vee r) = (p \vee q) \vee r$
- $p \wedge (q \wedge r) = (p \wedge q) \wedge r$
- $p \vee q = q \vee p$
- $p \wedge q = q \wedge p$
- $p \vee (p \wedge q) = p$
- $p \wedge (p \vee q) = p$
- $p \vee \mathsf{true} = \mathsf{true}$
- $p \vee \mathsf{false} = p$
- $(p \vee (r \wedge q) = p) \;\Rightarrow\; ((q \Rightarrow p) \vee r = (q \Rightarrow p))$
- $((q \Rightarrow p) \vee r = (q \Rightarrow p)) \;\Rightarrow\; (p \vee (r \wedge q) = p)$
- $\neg p = (p \Rightarrow \mathsf{false})$

Models of HA are called *Heyting algebras*.

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

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

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

2.7.2 Reachable semantics

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

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

$$
\begin{array}{lll}
\textbf{spec}\ \textsc{SetNat} = & \textbf{sorts} & \mathit{bool}, \mathit{nat}, \mathit{set} \\
& \textbf{ops} & \mathit{true}{:}\,\mathit{bool} \\
& & \mathit{false}{:}\,\mathit{bool} \\
& & \_\_ \vee \_\_{:}\,\mathit{bool} \times \mathit{bool} \to \mathit{bool} \\
& & 0{:}\,\mathit{nat} \\
& & \mathit{succ}{:}\,\mathit{nat} \to \mathit{nat} \\
& & \mathit{eq}{:}\,\mathit{nat} \times \mathit{nat} \to \mathit{bool} \\
& & \varnothing{:}\,\mathit{set} \\
& & \mathit{add}{:}\,\mathit{nat} \times \mathit{set} \to \mathit{set} \\
& & \_\_ \in \_\_{:}\,\mathit{nat} \times \mathit{set} \to \mathit{bool} \\
& \multicolumn{2}{l}{\forall p{:}\,\mathit{bool}, m, n{:}\,\mathit{nat}, S{:}\,\mathit{set}} \\
& & \bullet\ p \vee \mathit{true} = \mathit{true} \\
& & \bullet\ p \vee \mathit{false} = p \\
& & \bullet\ \mathit{eq}(n,n) = \mathit{true} \\
& & \bullet\ \mathit{eq}(0, \mathit{succ}(n)) = \mathit{false} \\
& & \bullet\ \mathit{eq}(\mathit{succ}(n), 0) = \mathit{false} \\
& & \bullet\ \mathit{eq}(\mathit{succ}(m), \mathit{succ}(n)) = \mathit{eq}(m,n) \\
& & \bullet\ n \in \varnothing = \mathit{false} \\
& & \bullet\ m \in \mathit{add}(n,S) = \mathit{eq}(m,n) \vee m \in S
\end{array}
$$

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

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

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

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

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

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

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

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

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

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

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

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

The equational calculus is sound for reasoning about the reachable standard models of presentations, since $RMod[\langle\Sigma,\Phi\rangle] \subseteq Mod[\langle\Sigma,\Phi\rangle]$ for any presentation $\langle\Sigma,\Phi\rangle$. It is sound to add induction rule schemes such as those given in Section 2.5; these are sound for any class of reachable models. Completeness is unachievable, for exactly the same reason as in the case of initial semantics; the proof of Theorem 2.5.26 can be repeated in this context almost without change. Finally, the techniques of term rewriting presented in Section 2.6 remain sound.

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

Example 2.7.11. Consider the following specification:

$$\begin{array}{lll} \textbf{spec}\ \mathrm{S{\small TATUS}} = & \textbf{sorts} & \mathit{status} \\ & \textbf{ops} & \mathit{single}{:}\mathit{status} \\ & & \mathit{married}{:}\mathit{status} \\ & & \mathit{widowed}{:}\mathit{status} \\ & \multicolumn{2}{l}{\bullet\ \mathit{widowed} = \mathit{single} \vee \mathit{widowed} = \mathit{married}} \end{array}$$

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

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

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

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

Another alternative to initial semantics deserves brief mention.

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

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

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

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

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

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

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

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

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

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

2.7.3 Dealing with partial functions: error algebras

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

Error algebras (this subsection): Predecessor is regarded as a total function, with $pred(0)$ specified to yield an *error* value.
Partial algebras (Section 2.7.4): Predecessor is regarded as a partial function.
Order-sorted algebras (Section 2.7.5): Predecessor is regarded as a total function on a sub-domain that excludes the value 0.

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

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

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

$$
\begin{array}{lll}
\textbf{spec}\ \textsc{NatPred} = & \textbf{sorts} & nat \\
& \textbf{ops} & 0{:}\,nat \\
& & succ{:}\,nat \to nat \\
& & pred{:}\,nat \to nat \\
& & error{:}\,nat \\
& & \_\_+\_\_{:}\,nat \times nat \to nat \\
& & \_\_\times\_\_{:}\,nat \times nat \to nat \\
& \multicolumn{2}{l}{\forall m,n{:}\,nat} \\
& & \bullet\ pred(succ(n)) = n \\
& & \bullet\ pred(0) = error \\
& & \bullet\ 0+n = n \\
& & \bullet\ succ(m)+n = succ(m+n) \\
& & \bullet\ 0 \times n = 0 \\
& & \bullet\ succ(m) \times n = (m \times n) + n
\end{array}
$$

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

$$
\begin{array}{lll}
\textbf{spec}\ \textsc{NatPred} = & \textbf{sorts} & nat \\
& \textbf{ops} & \ldots \\
& \multicolumn{2}{l}{\forall m,n{:}\,nat} \\
& & \bullet\ \ldots \\
& & \bullet\ succ(error) = error \\
& & \bullet\ pred(error) = error \\
& & \bullet\ error + n = error \\
& & \bullet\ n + error = error \\
& & \bullet\ error \times n = error \\
& & \bullet\ n \times error = error
\end{array}
$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$
\begin{array}{lll}
\textbf{spec } \text{NATPRED} = & \textbf{sorts} & nat \\
& \textbf{ops} & 0{:}\, nat \\
& & succ{:}\, nat \to nat \\
& & pred{:}\, nat \to nat \qquad\qquad :unsafe \\
& & error{:}\, nat \qquad\qquad\quad\;\; :unsafe \\
& & \_\_+\_\_{:}\, nat \times nat \to nat \\
& & \_\_\times\_\_{:}\, nat \times nat \to nat \\
& \multicolumn{2}{l}{\forall m, n{:}\, nat} \\
& & \bullet\ pred(succ(n)) = n \\
& & \bullet\ pred(0) = error \\
& & \bullet\ 0 + n = n \\
& & \bullet\ succ(m) + n = succ(m + n) \\
& & \bullet\ 0 \times n = 0 \\
& & \bullet\ succ(m) \times n = (m \times n) + n
\end{array}
$$

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

$$\begin{array}{ll} \textbf{spec } \text{NATPRED} = & \textbf{sorts } nat \\ & \textbf{ops } \ \ldots \\ & \forall m,n{:}nat, k{:}nat{:}unsafe \\ & \quad \bullet \ \ldots \\ & \quad \bullet \ pred(error) = error \\ & \quad \bullet \ succ(error) = error \\ & \quad \bullet \ error + k = error \\ & \quad \bullet \ k + error = error \\ & \quad \bullet \ error \times k = error \\ & \quad \bullet \ k \times error = error \end{array}$$

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

$$\begin{array}{ll} \textbf{spec } \text{NATPRED} = & \textbf{sorts } nat \\ & \textbf{ops } \ \ldots \\ & \qquad recover{:} nat \to nat \\ & \forall m,n{:}nat, k{:}nat{:}unsafe \\ & \quad \bullet \ \ldots \\ & \quad \bullet \ recover(error) = 0 \\ & \quad \bullet \ recover(n) = n \end{array}$$

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

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

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

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

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

$$\begin{array}{ll} \textbf{spec } \text{BOUNDEDNAT} = & \textbf{sorts } nat \\ & \textbf{ops } \ 0{:}nat \\ & \qquad succ{:}nat \to nat \ {:}unsafe \\ & \qquad overflow{:}nat \quad {:}unsafe \\ & \bullet \ succ(succ(succ(succ(succ(succ(0)))))) = overflow \end{array}$$

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

2.7.4 Dealing with partial functions: partial algebras

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

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

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

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

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

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

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

If moreover h satisfies the condition

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

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

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

Exercise 2.7.32. Consider a partial Σ-algebra A and its underlying total Σ-algebra $A_\perp$. Given any Σ-congruence $\equiv$ on $A_\perp$, removing all pairs involving $\perp$ yields a *strong Σ-congruence on A*. Check that such strong congruences are exactly kernels of strong Σ-homomorphisms, cf. Exercises 1.3.14 and 1.3.18. Check that strong congruences are equivalence relations that preserve and reflect definedness of operations and are closed under defined operations. Kernels of weak Σ-homomorphisms are *weak Σ-congruences*: equivalence relations that are closed under defined operations. Spell out these definitions in detail. For any partial Σ-algebra A and weak Σ-congruence $\equiv$ on A, generalise Definition 1.3.15 to define the *quotient of A by $\equiv$*, written $A/\!\equiv$. Note that an operation is defined in $A/\!\equiv$ on a tuple of equivalence classes provided that in A it is defined on at least one tuple of their respective elements. Check which of Exercises 1.3.18–1.3.23 carry over. □

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

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

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

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

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

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

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

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

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

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

Exercise 2.7.37. Complete the above proof by showing that:

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

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

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

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

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

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

```
spec NatPred = sorts nat
               ops   0: nat
                     succ: nat → nat
                     pred: nat → nat
                     __+__: nat × nat → nat
                     __×__: nat × nat → nat
               ∀m, n: nat
                     • def(0)
                     • def(succ(n))
                     • pred(succ(n)) = n
                     • 0 + n = n
                     • succ(m) + n = succ(m + n)
                     • 0 × n = 0
                     • succ(m) × n = (m × n) + n
```

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

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

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

2.7.5 Partial functions: order-sorted algebras

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Exercise 2.7.50. Define term evaluation. □

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

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

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

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

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

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

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

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

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

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

$$
\begin{array}{lll}
\textbf{spec}\ \textsc{NatPred} = & \textbf{sorts} & nznat \le nat \\
& \textbf{ops} & \ldots \\
& & \_\_+\_\_ : nznat \times nat \to nznat \\
& & \_\_+\_\_ : nat \times nznat \to nznat \\
& & \_\_\times\_\_ : nznat \times nznat \to nznat \\
& \ldots &
\end{array}
$$

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

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

$$
\begin{array}{lll}
\textbf{spec}\ \textsc{NatPred} = & \textbf{sorts} & nznat \le nat \\
& \textbf{ops} & \ldots \\
& & r: nat \to nznat \\
& \multicolumn{2}{l}{\forall m,n{:}\,nat, k{:}\,nznat} \\
& & \bullet\ \ldots \\
& & \bullet\ r(n) = n
\end{array}
$$

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

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

$$
\begin{array}{lll}
\textbf{spec } \mathrm{N{\scriptsize AT}P{\scriptsize RED}} = & \textbf{sorts} & nat \le nat? \\
& \textbf{ops} & 0{:}\, nat \\
& & succ{:}\, nat \to nat \\
& & pred{:}\, nat \to nat? \\
& & \_\_+\_\_{:}\, nat \times nat \to nat \\
& & \_\_\times\_\_{:}\, nat \times nat \to nat \\
& \forall m,n{:}\, nat & \\
& & \bullet\ pred(succ(n)) = n \\
& & \bullet\ 0+n = n \\
& & \bullet\ succ(m)+n = succ(m+n) \\
& & \bullet\ 0 \times n = 0 \\
& & \bullet\ succ(m) \times n = (m \times n)+n
\end{array}
$$

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

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

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

2.7.6 Other options

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

Example 2.7.55 (First-order predicate logic). Signatures may be modified to enable them to include (typed) *predicate names* in addition to operation names, e.g. $\_\_\le\_\_{:}\, nat \times nat$. Atomic formulae are then formed by applying predicates to terms; in *first-order predicate logic with equality*, the predicate $\_\_=\_\_{:}\, s \times s$ is implicitly available for any sort s. Formulae are built from atomic formulae using logical connectives and quantifiers. Algebras are modified to include relations on their carriers to interpret predicate names; the interpretation of the built-in equality predicate (if available) may be forced to be the underlying equality on values, or it may merely be required to be a congruence relation. Homomorphisms are required to respect predicates as well as operations. The satisfaction of a *sentence* (a formula without free variables) by an algebra is as usual in first-order logic. See Example 4.1.12 for details of the version of first-order predicate logic with equality we will use. Presentations involving predicates and first-order axioms are appropriate for the specification of programs in *logic programming languages* such as Prolog, where the Horn clause fragment of first-order logic is used for writing the programs themselves. Note that such presentations may have no models at all, but even if they have some models, they may have no initial models (see Example 2.7.11) or no final models (see Exercise 2.7.14), or even no reachable models. (**Exercise:** Give a specification with first-order axioms having some models but no reachable model.) □

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

Example 2.7.57 (Polymorphic types). Programming languages such as Standard ML [Pau96] can be used to define *polymorphic types* such as α *list* (instances of which include *bool list* and (*bool list*) *list*) and *polymorphic values* such as $head{:}\forall\alpha \bullet \alpha\, list \rightarrow \alpha$ (which is then applicable to values of types such as *bool list* and (*bool list*) *list*). To specify such types and functions, signatures are modified to contain *type constructors* in place of sort names; for example, *list* is a unary type constructor and *bool* is a nullary type constructor. Terms built using these type constructors and *type variables* (such as α above) are the *polymorphic types* of the signature. The set Ω of operation names is then indexed by non-empty sequences of polymorphic types, where $f \in \Omega_{t_1\dots t_n,t}$ means $f{:}\forall FV(t_1) \cup \ldots \cup FV(t_n) \cup FV(t) \bullet t_1 \times \cdots \times t_n \rightarrow t$. There are various choices for algebras over such signatures. Perhaps the most straightforward choice is to require each algebra A to incorporate a (single-sorted) *algebra of carriers* $Carr(A)$, having sets interpreting types as values and an operation to interpret each type constructor. Then, for each operation $f \in \Omega_{t_1\dots t_n,t}$ and for each instantiation of type variables $i{:}V \rightarrow |Carr(A)|$, A has to provide a function $f_{A,i}{:}\, i^\#(t_1) \times \cdots \times i^\#(t_n) \rightarrow i^\#(t)$. Various conditions may be imposed to ensure that the interpretation of polymorphic operations is *parametric* in the sense of [Str67], by requiring $f_{A,i}$ and $f_{A,i'}$ to be appropriately related for different type variable instantiations i, i'. Another choice would be to interpret each type as the set of equivalence classes of a *partial equivalence relation* on a model of the untyped λ-calculus [BC88]. Axioms contain (universal) quantifiers for type variables in addition to quantifiers for ordinary variables, as in System F [Gir89]; alternatively, type variable quantification may be left implicit, as in Extended ML [KST97]. □

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

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

2.8 Bibliographical remarks

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

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

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

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

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

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

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

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

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

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

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

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

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

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

First-order predicate logic has been used as a framework for algebraic specification in various approaches, see for instance CIP-L [BBB$^+$85] and CASL [Mos04]. See [Poi86], [MTW88], [Mei92] and [Qia93] for different approaches to the algebraic specification of higher-order functions. Frameworks that cater for the specification of polymorphic types and functions are described in [MSS90], [Mos89] and [KST97]. See [Nip86] for more on algebras with non-deterministic operations; for a different approach using relation algebra, see [BS93]. See [WM97] for a comprehensive overview. Soundness and completeness of term rewriting for non-deterministic specifications is studied in [Hus92]. Continuous algebras and the use of Scott-style domain-theoretic techniques in algebraic specification were first discussed in [GTWW77]. See [Sch86] or [GS90] for much more on domain theory itself. Although these and other extensions to the standard framework have been explored separately, the few attempts that have been made to combine such extensions (see e.g. [AC89] and [Mos04]) have tended to reveal new problems.

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.