INSTITUTIONS: Abstract Model Theory for Specification and Programming*

Joseph A. Goguen† Rod Burstall‡

Abstract

There is a population explosion among the logical systems used in Computing Science. Examples include first order logic, equational logic, Horn clause logic, higher order logic, infinitary logic, dynamic logic, intuitionistic logic, order-sorted logic, and temporal logic; moreover, there is a tendency for each theorem prover to have its own idiosyncratic logical system. We introduce the concept of *institution* to formalise the informal notion of "logical system." The major requirement is that there is a satisfaction relation between models and sentences which is consistent under change of notation. Institutions enable us to abstract from syntactic and semantic detail when working on language structure "in-the-large"; for example, we can define language features for building large structures from smaller ones, possibly involving parameters, without commitment to any particular logical system. This applies to both specification languages and programming languages. Institutions also have applications to such areas as database theory and the semantics of artificial and natural languages. A first main result of this paper says that any institution such that signatures (which define notation) can be glued together, also allows gluing together theories (which are just collections of sentences over a fixed signature). A second main result considers when theory structuring is preserved by institution morphisms. A third main result gives conditions under which it is sound to use a theorem prover for one institution on theories from another. A fourth main result shows how to extend institutions so that their theories may include, in addition to the original sentences, various kinds of constraint that are useful for defining abstract data types, including both "data" and "hierarchy" constraints. Further results show how to define institutions that allow sentences

*Research supported in part by grants from the Science and Engineering Research Council, the National Science Foundation, and the System Development Foundation, plus Office of Naval Research contracts N00014-80-0296, N00014-82-C-0333, N00014-85-C-0417 and N00014-86-C-0450.

†Programming Research Group, Oxford University, and SRI International, Menlo Park, California.

‡Computer Science department, University of Edinburgh.

and constraints from two or more institutions. All our general results apply to such "duplex" or "multiplex" institutions.

1 Introduction

Recent work in theoretical Computing Science uses many different logical systems. Perhaps most popular are the many variants of first and higher order logic found in current generation theorem provers. But also popular are equational logic, as used to study abstract data types, and Horn clause logic, as used in "logic programming," e.g., Prolog. More exotic logical systems, such as temporal logic, second order polymorphic lambda calculus, dynamic logic, order-sorted logic, modal logic, continuous algebra, infinitary logic, intuitionistic higher order type theory, and intensional logic have been proposed to handle problems such as concurrency, overloading, exceptions, non-termination, program construction, and natural language. However, it seems apparent that many general results used in the applications are actually *completely independent* of what underlying logic is chosen. In particular, if we are correct that the essential purpose of a specification language is to say how to put (hopefully small and well-understood) theories together to make new (and possibly very large) specifications [15], then much of the syntax and semantics of specification does not depend upon the logical system in which the theories are expressed; similar considerations apply to implementing a specification, verifying correctness, and programming-in-the-large. Also, because of the proliferation of logics of programming and of logic-based programming languages, plus the great expense of implementing tools like theorem provers and compilers, it is useful to know when sentences in one logic can be translated into sentences in another logic in such a way that soundness is preserved. This will allow, for example, using a theorem prover for one logic on (translations of) sentences from another logic, or using a compiler for one logic-based language on (translations of) programs from another. Institutions provide a foundation for approaching these and many other problems in Computing Science.

One of the most essential elements of a logical system is its relationship of *satisfaction* between its *syntax* (i.e., its sentences) and its *semantics* (i.e., its models); this relationship is sometimes called a "model theory" and classically appear in the form of a Galois connection (as in Section 2.2 below). Whereas traditional model theory assumes a fixed vocabulary, institutions allow us to consider many different vocabularies at once. Informally, an institution consists of

- a collection of signatures (which are vocabularies for use in constructing sentences in a logical system) and signature morphisms, together

with for each signature Σ,

- a collection of Σ-sentences,
- a collection of Σ-models, and
- a Σ-satisfaction relation, of Σ-sentences by Σ-models,

such that when you change signatures (by a signature morphism), satisfaction of sentences by models changes consistently.

The first main result in this paper (Theorem 11) states that any institution whose declarations of notation (as given by signatures) can be glued together will also allow gluing together *theories* (which are collections of sentences) to form larger specifications, using colimits. A second main result (Proposition 23) states that any institution extends to another whose sentences may be either the old sentences, or else new "data constraints" which capture induction and are useful in defining abstract data types. Theorem 26 extends this to more general kinds of constraint, including so-called "hierarchy constraints," and Theorem 24 says that colimits extend from signatures of the original institution to theories over the extended institution. Another result (Proposition 35) shows that a suitable *institution morphism* permits a theorem prover for one institution to be used on theories from another. Another main result (Theorem 39) shows that many institution morphisms preserve structuring operations on theories. This implies that constructions for programming-in-the-large are preserved by certain language translations. Again using the notion of institution morphism, "duplex" institutions permit combining sentences from one institution with constraints from another (Theorem 44); moreover, the duplex institution again inherits colimits from the signature category of the base institution (Theorem 45). Finally, "multiplex" institutions permit whatever combination of sentences and constraints one might desire, provided they are related by morphisms to the same base institution (Section 4.3). Altogether, this gives a very rich and flexible framework that can be used in program specification and logical programming, as well as many other other areas of Computing Science.

Institutions arose in our research on the specification language Clear [15] under the rather general name of "language" [17]. The present paper adds many new concepts, results, and examples, as well as an improved notation. Barwise's approach to "abstract model theory" [6] (see also [5]) resembles our work in its intention to generalise basic results in model theory and in its use of elementary category theory; but [5] is more concrete (for example, its syntactic structures are limited to the usual function, relation and logical symbols) and is focused on classical results of logic.

Section 2 below gives the basic definitions and results for institutions and theories. Section 3 discusses constraints, while Section 4 considers the use of two or more institutions. Appendix A.1 briefly reviews universal algebra, emphasising the results needed to show that equational logic is indeed an institution, while Appendix A.2 considers the (many-sorted) first order, first order with equality, Horn clause, and conditional equational institutions. Some readers may wish to read these appendices in parallel with Section 2.

1.1 Methodology and Logical Systems

Systematic program design requires a careful specification of the problem to be solved. But experience in software engineering shows that there are major difficulties in producing consistent, rigorous specifications that adequately reflect users' requirements for complex systems. We suggest that these difficulties can be ameliorated by making specifications as modular as possible, so that they are built from small, understandable pieces, many of which may be used repeatedly (for example, those defining concepts like "ordering," "list," or "file"). Modern work in programming methodology supports the view that abstraction, and in particular data abstraction, is a useful way to obtain such modularity, and that parameterised (also called "generic") specifications can lead to further improvements.

One way to apply these ideas is through a specification language that supports putting together parameterised abstractions. Whereas a specification written directly in a logical system is an unstructured, and possibly unmanageably large, collection of sentences, a suitable specification language can make it much easier to write and to read specifications, especially for large systems. Specification languages that support modularity in one way or another include Clear [15, 17, 18], OBJ [32, 56, 48, 33], Z [1, 61, 96], Act One [25], Act Two [29], ASL [105, 89], and Extended ML [87]. We also suggest that modularity may be useful in proving theorems about specifications, for example, in proving that a given program actually meets its specification. Moreover, exactly the same structuring mechanisms can be used to achieve exactly the same advantages in programming languages that are rigorously based upon some formal system of logic; we shall call such languages *logical programming languages*. Finally, the same techniques of modularisation can be applied to conventional imperative programming languages such as Ada, by using a system like LIL [39, 104].

In order for a specification written in a given language to have a precise meaning, it is necessary for that language to have a precise semantics. (This may seem obvious, but the fact is that many specification languages lack such a semantics.) Part of that semantics will be an underlying logical system, which must have certain properties to be useful for this task.

These include suitable notions of model and sentence, and a satisfaction relationship between sentences and models that is invariant under change of notation. For applications to programming, we will want to define models by programs.

Signature morphisms play a basic role in structuring specifications. Let us assume for concreteness of exposition that the signatures have sorts and operators, and then consider some specific structuring mechanisms. First, we may build a more complex specification by adding new sorts and operators to an existing signature; then the inclusion of the original signature into the extended signature is an "enrichment" signature morphism. Second, we may wish to use such an enrichment not just on one specification, but on a whole class of specifications. This leads to parameterised specifications. For instantiation, the parameter sorts and operators are bound to particular sorts and operators by a "binding" signature morphism. Third, a large specification may have name clashes: two subspecifications may happen to use the same sort or operator names. These can be eliminated by signature morphisms that define renamings. Enrichment, binding and renaming raise no deep logical problems, but are still important for modular structure. Using institutions, we can define such features without making a commitment to any particular logical system. Moreover, the task of giving a semantics for the language is also simplified. We feel that these considerations justify an attempt to deal with logical systems in a general way, free of the entanglements of any particular syntax and semantics.

A specification language is not a programming language. Thus, the denotation of an Algol text is one or more function, but the denotation of a specification text is a *theory*, that is, a collection of sentences *about* programs. Of course, a theory also has a denotation, which is the collection of all models that satisfy the sentences in the theory, and this should be taken as the ultimate denotation of a specification text. Programmers construct such models using programming languages. For a (pure) logical programming language, the specification is also a program, so in this case there is no need to verify that these two agree.

Despite this distinction, it may be useful to view a specification language like Clear [15, 17, 18] as a functional language with types. Its values are specifications, its functions are specification constructing operations, and its types denote classes of specifications. Because theories are used for specifications and theory morphisms for bindings, we have the following correspondence:

value	theory
function	theory morphism
binding	theory morphism
application of a function to a value	pushout of theory morphisms
type	theory
values of type T	theories with a morphism from T

Clear was designed to work with any institution. To provide initiality for abstract data types, we can further elaborate values to be "theories with duplex constraints," as discussed in Section 4.2. This elaboration makes sense for any institution, and the elaborated theories also form an institution.

1.2 Related Work and Applications

Although this paper is a direct descendent of the first paper on institutions [44], delays in publication have led to the situation that the literature on institutions and their applications is now so large that it would be awkward to undertake more than the following brief survey of some representative papers and results.

We begin with some of our own work. The institution concept was introduced in [17] to help define the semantics of Clear [15]. Thus, Clear can be used to build large specifications from small, reusable theories over any logical system. Clear's approach to modularity is called *parameterised programming* [38]. Parameterised programming is implemented in OBJ [32, 56, 48, 33], which can be considered an implementation of Clear for the institution of order sorted equational logic [53], and it has also been used in designing the logical programming languages Eqlog [51] and FOOPS [52], the latter of which is object-oriented. Parameterised programming can even be applied to a conventional imperative programming language like Ada, where it provides module interconnection capabilities far beyond those in the language itself; see LIL [39, 104]. Each of these language designs relies on the machinery in this paper, especially Theorems 11 and 24, but instantiated with different institutions.

In [47] we developed so-called "charters" and "parchments" as easier ways to generate institutions, exploiting the fact that the syntax of a logical system forms an initial algebra. In particular, the difficulty of checking the Satisfaction Condition (in Definition 1 below) is avoided, because this condition is automatically satisfied. A quite different formalisation of the intuitive notion of logical system, axiomatising the category of theories as well as that of signatures, is given in [45].

Other researchers have produced much interesting work. Mayoh [76] proposed generalising institutions so that satisfaction is no longer just Boolean-

valued, and pointed out that this generalisation would have applications to database systems, where the answer to a given query (considered as a sentence) for a given model might be some complex proposition, or set of values, rather than just *true* or *false*. This generalisation is somewhat further developed in [47]. Mayoh [76] also suggested some applications to the semantics of programming and natural languages.

Two other specification languages based on institutions are ASL [89] and Extended ML [87], both due to Sannella and Tarlecki. Extended ML is integrated with the widely used functional programming language ML [59]. A number of topics in theoretical Computing Science have been abstracted to the level of institutions, including observational equivalence by Tarlecki and Sannella [88] and data representations (usually called "implementations" in the abstract data type literature) by Beierle and Voss [7, 87]. Tarlecki has also studied free constructions in institutions [98, 99], and moreover has shown that certain model theoretic results generalise to institutions, including equivalence of the Craig Interpolation and the Robinson Consistency Theorem, under certain assumptions [97]. Probably much more could be done along these lines. Lowry has applied institutions to problem reformulation [72], and Mosses has used institutions in his elegant development of unified algebra [80] for the denotational semantics of programming languages. The machinery of the present paper has also influenced work that does not use institutions directly. In particular, our abstract notion of constraint (Section 3) has influenced many authors in the context of algebraic specification, that is, for the equational institution, e.g., [28, 81, 24]. The second volume of Ehrig and Mahr's book [27] contains an extensive discussion of constraints for the algebraic case, and also some results in an institutional context.

Several authors have felt the need for a formalisation of logical systems that is based on deduction rather than satisfaction. Fiadeiro and Sernadas [30] introduce "π-institutions," based on a consequence relation like that of Tarski's deductive systems [101], and show that the main results of the present paper also hold in that setting; [30] also discusses some applications to conceptual modelling and knowledge representation, as do [31] and [93], the latter also mentioning some interesting applications to database theory. Meseguer [78] provides a general approach to logical systems which includes axiomatisations of the notions of entailment system (building on Dana Scott's axiomatisation of deduction [91]) and proof calculus, as well as institution. This avoids commitment to any particular style of proof theory, and in particular can handle effective (i.e., computable) proof calculi. Meseguer [78] also provides general notions of morphism for each level, as well as a theory of categorical logics, including some important basic results, such as

a very general structure-semantics adjointness; our approach to categorical logic (in which proofs are morphisms, as advocated by Lambek and (Phil) Scott [66]) is given in Section 2.5.1. Meseguer also axiomatises the notion of logical programming language in a way that uses most of the concepts in his paper.

Logical frameworks [58, 4] are another formalism for defining logics at the same level of generality as institutions. Harper, Sannella and Tarlecki [60] define a "logical system" as a family of consequence relations indexed by signatures, obtaining a notion equivalent to that of Fiadeiro and Sernadas, but applied to logical frameworks. Poigné [83] studies "foundations," where the set of sentences associated with a signature becomes a category indexed by sets of variables; this gives a double indexing, over the constants in the signature and over variables. The extra complexity may or may not be worthwhile.

1.3 Prerequisites

Although relatively little category theory is needed for most of this paper, we have not resisted the temptation to add some more arcane remarks for those who may be interested. We must assume the reader is already acquainted with the notions of category, functor and natural transformation. Occasional remarks use adjoint functors. There are several introductions to these ideas, including [19, 2, 57], and for the mathematically more sophisticated [67]. Familiarity with the initial algebra approach to abstract data types is helpful, but probably not necessary; see [54] or [26]. Colimits are briefly explained in Section 2.4. A general motivation for the use of category theory in Computing Science is given in [41].

By way of notation, categories are boldface, $|\mathbf{C}|$ denotes the class of objects of $\mathbf{C}$, $f;g$ denotes the composition of morphisms f and g in diagrammatic order, 1_A denotes the identity at an object A, $\cong$ denotes isomorphism, and $\mathbf{C}^{op}$ denotes the opposite category of $\mathbf{C}$ (see [67], II.2). The notation that we use for general algebra is developed in Section A.1, while Section A.2 develops first order logic.

1.4 Acknowledgements

We wish to thank the following institutions for their support at various times while this research was being conducted: the National Science Foundation; the Science and Engineering Research Council of Great Britain; the Office of Naval Research; the Center for the Study of Language and Information at Stanford University; and a British Petroleum Venture Research Fellowship. We wish to thank Eleanor Kerse for typing some early drafts, and

José Meseguer for his extensive comments. Andrzej Tarlecki also made very detailed and valuable comments on two drafts of the paper during a visit to CSLI in early 1985. We are grateful to many people for helpful conversations and suggestions, notably our ADJ collaborators, Jim Thatcher, Eric Wagner and Jesse Wright, and also Peter Dybjer, Gordon Plotkin, David Rydeheard, John Reynolds, Don Sannella, and Steve Zilles. We thank Răzvan Diaconescu and Hendrik Hilberdink for help with Theorem 31 and Proposition 37. Special thanks to Kathleen Goguen and Seija-Leena Burstall for extreme patience, and to the Venerable Chögyam Trungpa Rinpoche for general inspiration.

2 Institutions

An institution consists of a category of signatures such that associated with each signature are sentences, models, and a relationship of satisfaction that, in a certain sense, is invariant under change of signature. Two familiar examples of this setup are equational logic (also called general, or universal, algebra) and first order logic (or model theory). In equational logic, a signature Σ declares the function symbols that are available, Σ-sentences are equations using these function symbols, and Σ-models are Σ-algebras. In first order logic, signatures in addition give relation symbols, sentences are the usual first order sentences, and models are the usual first order structures. In both cases, satisfaction is the familiar relation. Appendices A.1 and A.2 discuss the many-sorted variants of these two examples in some detail.

2.1 Definition and Examples

The essence of the institution notion is that a change of signature (by a signature morphism) induces "consistent" changes in sentences and models, in a sense made precise by the "Satisfaction Condition" in Definition 1 below. This goes a step beyond Tarski's classic "semantic definition of truth" [102], and also generalises Barwise's "Translation Axiom" [6]. The wide range of consequences, and the fact that even for equational logic, the Satisfaction Condition is not entirely trivial, suggest that this step has some substance. Moreover, it is a basic and familiar fact that the truth of a sentence (in logic) is independent of the symbols chosen to represent its functions and relations. This can be summed up in the slogan

Truth is invariant under change of notation.

It is also fundamental that sentences translate in the same direction as the change of notation, whereas models translate in the opposite direction. Because reversing the direction of morphisms gives a contravariant functor, the definition below uses $\mathbf{Cat}^{op}$, the opposite of the category of categories.

Definition 1: An **institution** $\mathcal{I}$ consists of

1. a category **Sign**, whose objects are called **signatures**,
2. a functor $Sen\colon\ \mathbf{Sign} \to \mathbf{Set}$, giving for each signature a set whose elements are called **sentences** over that signature,
3. a functor[1] $\mathbf{Mod}\colon\ \mathbf{Sign} \to \mathbf{Cat}^{op}$ giving for each signature Σ a category whose objects are called Σ-**models**, and whose arrows are called Σ-(model) **morphisms**, and
4. a relation $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times Sen(\Sigma)$ for each $\Sigma \in |\mathbf{Sign}|$, called Σ-**satisfaction**,

such that for each morphism $\phi\colon\ \Sigma \to \Sigma'$ in **Sign**, the **Satisfaction Condition**

$$m' \models_{\Sigma'} Sen(\phi)(e) \quad \text{iff} \quad \mathbf{Mod}(\phi)(m') \models_\Sigma e$$

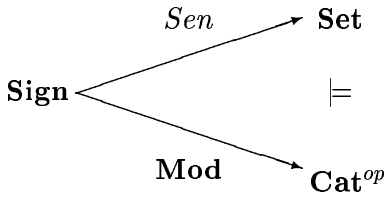
holds for each $m' \in |\mathbf{Mod}(\Sigma')|$ and each $e \in Sen(\Sigma)$. We will write $\phi(e)$ or even ϕe for $Sen(\phi)(e)$, and $\phi(m')$ or $\phi m'$ for $\mathbf{Mod}(\phi)(m')$; also we will drop the signature subscripts on the satisfaction relation when it is not confusing. □

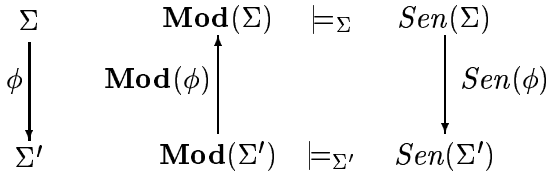
These conventions are used in the following condensed form of the Satisfaction Condition,

$$m' \models \phi e \quad \text{iff} \quad \phi m' \models e.$$

The following picture may help in visualising these relationships:

[1] Here, and at other places in this paper, some readers may have questions about set-theoretic foundations, because **Cat** clearly needs to include "large" categories. In fact, we stay well away from anything genuinely problematical, and nearly any foundation that has been proposed for category theory will do, in particular, the "hierarchy of universes" discussed e.g., by MacLane [67], in Section I.6.





Appendix A of this paper shows that a number of logical systems satisfy Definition 1, including (the many sorted versions of) equational logic, first order logic, Horn clause logic, Horn clause logic with equality, and first order logic with equality. We note that it can be non-trivial to establish the Satisfaction Condition.

For some purposes, Definition 1 can be simplified by replacing $\mathbf{Mod}: \mathbf{Sign} \to \mathbf{Cat}^{op}$ by a functor $Mod: \mathbf{Sign} \to \mathbf{Set}^{op}$, where $Mod(\Sigma)$ is the collection of all Σ-models; the two versions of the definition are thus related by the equation $Mod(\Sigma) = |\mathbf{Mod}(\Sigma)|$. Indeed, this was our original version [17]. Some reasons for changing it are: first, it is more consistent with the categorical point of view to consider morphisms of models along with models; and second, we want every liberal institution to be an institution, rather than just to determine one (liberal institutions are discussed later in this paper). Section 2.5 gives a more categorical definition of institutions which replaces the perhaps *ad hoc* looking family of satisfaction relations by a functor into a category of "twisted relations." This more categorical formulation suggests some further generalisations of the institution concept, including one which models deductions as morphisms among sentences.

2.2 Theories and Theory Morphisms

If, as suggested in the introduction, a *specification* provides a mathematical theory of the intended behaviour of a program, and if a *theory* consists of all the sentences that are true of that behaviour, then it will be important to define and develop the basic properties of theories over an arbitrary in-

stitution. A theory consists of a signature Σ and a "closed" collection of Σ-sentences, as in first order logic. Because these theories usually contain an infinity of sentences, one concern is to define them by a finite subset. This notion differs from the "algebraic theories" of Lawvere [69], which are independent of choice of signature, and it simplifies the "signed theories" of [15].

Definition 2: Let $\mathcal{I}$ be a fixed but arbitrary institution. Then

1. A Σ-**presentation** is a pair $\langle \Sigma, E \rangle$, where Σ is a signature and E is a collection of Σ-sentences.

2. A Σ-model m **satisfies** a presentation $\langle \Sigma, E \rangle$ if it satisfies each sentence in E; write $m \models E$ in this case.

3. Given a collection E of Σ-sentences, let E^* be the collection of all Σ-models that satisfy each sentence in E.

4. Given a collection M of Σ-models, let M^* be the collection of all Σ-sentences that are satisfied by each model in M; also, let M^* denote $\langle \Sigma, M^* \rangle$, called the **theory** of M.

5. The **closure** of a collection E of Σ-sentences is E^{**}, denoted $E^\bullet$.

6. A collection E of Σ-sentences is **closed** iff $E = E^\bullet$.

7. A Σ-**theory** is a presentation $\langle \Sigma, E \rangle$ such that E is closed.

8. The Σ-theory **presented** by a presentation $\langle \Sigma, E \rangle$ is $\langle \Sigma, E^\bullet \rangle$.

9. A Σ-sentence e is **semantically entailed** by a collection E of Σ-sentences, written $E \models e$, iff $e \in E^\bullet$.

□

Our definition of closure is based on satisfaction rather than deduction. Of course, some institutions have a natural complete set of inference rules, for example, the many-sorted equational institution; but others do not, and we prefer the added generality.

We can also consider closed collections of models; following the terminology of the equational institution, these might be called **varieties**. It is often convenient to regard E^* as the full subcategory of $\mathbf{Mod}(\Sigma)$ with objects the models that satisfy E. The **closure** of a collection M of models is M^{**}, denoted $M^\bullet$, and a full subcategory of models is called **closed** iff its objects are exactly all the models of some collection of sentences.

Notice that there is a forgetful functor $Sign\colon \mathbf{Th} \to \mathbf{Sign}$ sending $\langle \Sigma, E \rangle$ to Σ, and sending ϕ as a theory morphism to ϕ as a signature morphism.

Proposition 3: The two functions denoted "*" in Definition 2 form what is known as a **Galois connection** (see, e.g., [21]), in that they satisfy the following properties, for any collections E, E' of Σ-sentences and collections M, M' of Σ-models:

1. $E \subseteq E'$ implies $E'^* \subseteq E^*$.
2. $M \subseteq M'$ implies $M'^* \subseteq M^*$.
3. $E \subseteq E^{**}$.
4. $M \subseteq M^{**}$.

These imply the following properties:

5. $E^* = E^{***}$.
6. $M^* = M^{***}$.
7. There is a dual (i.e., inclusion reversing) isomorphism between the closed collections of sentences and the closed collections of models; this isomorphism takes unions to intersections and intersections to unions.
8. $\bigcap_n E_n{}^* = (\bigcup_n E_n)^*$;
9. $(\bigcap_n E_n{}^*)^{**} = (\bigcup_n E_n)^*$;
10. $(\bigcup_n E_n{}^{**})^* = \bigcap_n E_n{}^*$;
11. $(\bigcup_n E_n{}^{**})^* = (\bigcup_n E_n)^*$;
12. $(\bigcap_n E_n{}^{**})^* = (\bigcup_n E_n{}^*)^{**}$.

There are also dual identities to 8.-12. for collections of models.

Proof: The first two assertions are straightforward. We now prove the third, assuming that $E \subseteq Sen(\Sigma)$: Because

$$E^* = \{m \mid (\forall e \in E)\, m \models e\},$$

then

$$\begin{aligned} E^{**} &= \{e' \mid (\forall m \in E^*)\, m \models e'\} \\ &= \{e' \mid (\forall m)\, m \in E^* \Rightarrow m \models e'\} \\ &= \{e' \mid (\forall m)\, [(\forall e \in E)\, m \models e] \Rightarrow m \models e'\}. \end{aligned}$$

But if e' is in E, and if $m \models e$ for all $e \in E$, then certainly $m \models e'$; thus the above set contains $\{e' \mid e' \in E\} = E$. The proof of the fourth assertion is similar, and the next eight assertions are familiar from lattice theory, e.g. [9]; in fact, 8.-12. follow easily from 7. □

The following identity is useful, for example, in Theorem 39 below:

Proposition 4: Given a signature morphism $\phi: \Sigma \to \Sigma'$ such that $\phi: \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ is surjective on objects[2] and a Σ-presentation E, then $\phi(\phi E)^* = E^*$.

Proof: Let $m \in |\mathbf{Mod}(\Sigma)|$. Then $m \in E^*$ iff $m \models E$, and $m \in \phi(\phi E)^*$ iff $[(\exists m' \in (\phi E)^*)\, m = \phi m']$ iff $[(\exists m')\, m' \models \phi E \;\&\; m = \phi m']$ iff $[(\exists m')\, m \models E \;\&\; m = \phi m']$ iff $[m \models E \;\&\; (\exists m')\, m = \phi m']$ iff $m \models E$ (because ϕ is surjective). □

Definition 5: If T and T' are theories, say $\langle \Sigma, E \rangle$ and $\langle \Sigma', E' \rangle$, then a **theory morphism** from T to T' is a signature morphism $\phi: \Sigma \to \Sigma'$ such that $\phi(e) \in E'$ for each $e \in E$; we will write $\phi: T \to T'$. The **category of theories** has theories as objects, and theory morphisms as morphisms, with composition and identities defined as for signature morphisms; we denote it **Th**. (It is easy to see that this is a category.) The following will frequently use capital letters to denote theory morphisms, e.g., $F: T \to T'$. □

For equational logic, there is another category with theories as objects, but with derivors, which map operators in Σ to derived operators in Σ', i.e., to Σ'-terms, as morphisms [46]; in fact, this kind of morphism agrees with the usual morphism notion for Lawvere theories [69].

For a theory T with signature Σ, let $\mathbf{Mod}(T)$ and also T^* denote the full subcategory of $\mathbf{Mod}(\Sigma)$ of all Σ-models that satisfy all the sentences in T.

Definition 6: Given a theory morphism $\phi: T \to T'$, the **forgetful functor** $\phi^*: T'^* \to T^*$ sends a T'-model m' to the T-model $\phi(m')$, and sends a T'-model morphism $f: m' \to n'$ to $\phi^*(f) = \mathbf{Mod}(\phi)(f): \phi(m') \to \phi(n')$. This functor is also denoted $\mathbf{Mod}(\phi)$. □

To ensure that this definition makes sense, we should check that if a given Σ'-model m' satisfies T', then $\phi^*(m')$ satisfies T. Let e be any sentence in T. Because ϕ is a theory morphism, $\phi(e)$ is a sentence of T' and therefore $m' \models \phi(e)$. The Satisfaction Condition now gives $\phi(m') \models e$, as desired. We also need that the morphism $\phi^*(f)$ lies in T^*, but this follows because T^* is a full subcategory of $\mathbf{Mod}(\Sigma)$, and the source and target objects of $\phi^*(f)$ lie in T^*.

[2] It does not matter what happens to the morphisms; in effect, this result concerns the notion of institution in which **Mod** takes values in $\mathbf{Set}^{op}$.

2.3 The Closure and Presentation Lemmas

Given a signature morphism $\phi\colon\ \Sigma \to \Sigma'$, a collection E of Σ-sentences, and a collection M' of Σ'-models, let us write $\phi(E)$ for $\{\phi(e) \mid e \in E\}$ and $\phi(M')$ for $\{\phi(m') \mid m' \in M'\}$. Given a collection M of Σ-models, let us also write $\phi^{-1}(M)$ for $\{m' \mid \phi(m') \in M\}$. Using this notation, we can write the Satisfaction Condition more compactly as

$$\phi^{-1}(E^*) = \phi(E)^*$$

and we also have

Lemma 7: Closure. $\phi(E^\bullet) \subseteq \phi(E)^\bullet$.

Proof: $\phi(E^{**})^* = \phi^{-1}(E^{***}) = \phi^{-1}(E^*) = \phi(E)^*$, using the Satisfaction Condition and 5. of Proposition 3. Therefore $\phi(E^\bullet) = \phi(E^{**}) \subseteq \phi(E^{**})^{**} = \phi(E)^{**} = \phi(E)^\bullet$, using 3. of Proposition 3 and the just proved equation. □

It is worth pointing out that the inclusion $\phi(E^\bullet) \subseteq (\phi E)^\bullet$ can be proper. This means that while $E \models e$ implies $\phi E \models \phi e$, the converse, that $\phi E \models \phi e$ implies $E \models e$, does *not* hold. The following gives an easier to check necessary and sufficient condition for a signature morphism to be a theory morphism.

Lemma 8: Presentation. Let $\phi\colon\ \Sigma \to \Sigma'$ and suppose that $\langle \Sigma, E\rangle$ and $\langle \Sigma', E'\rangle$ are presentations. Then $\phi\colon\ \langle \Sigma, E^\bullet\rangle \to \langle \Sigma', E'^\bullet\rangle$ is a theory morphism iff $\phi(E) \subseteq E'^\bullet$.

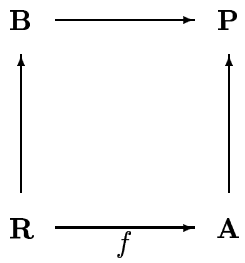
Proof: For the "if" part, we have by the Closure Lemma that $\phi(E^\bullet) \subseteq \phi(E)^\bullet$. By hypothesis, $\phi(E) \subseteq E'^\bullet$. Therefore, $\phi(E^\bullet) \subseteq \phi(E)^\bullet \subseteq E'^\bullet$, so ϕ is a theory morphism. Conversely, if ϕ is a theory morphism, then $\phi(E^\bullet) \subseteq E'^\bullet$. Thus $\phi(E) \subseteq E'^\bullet$ because $E \subseteq E^\bullet$. □

The Presentation Lemma tells us that to check whether ϕ is a theory morphism, we can apply ϕ to each sentence e of the source presentation E and see whether $\phi(e)$ is in the closure of E'; there is no need to check every sentence in $E^\bullet$.

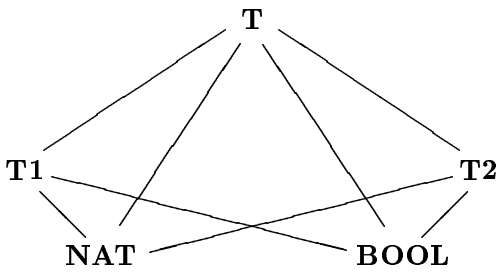
2.4 Putting Theories Together

The introduction of this paper suggests that a basic purpose of a specification language is to reduce the difficulty of describing large theories by providing mechanisms for constructing them from already available smaller theories. The specification language Clear [15, 16, 17] provided a number of such mechanisms that can be explicated using colimits in the category of theories

over an arbitrary institution. These include: the *sum* of theories (in such a way as not to duplicate any subtheories that may be shared; for example, **NAT** and **BOOL** might be subtheories of several theories, as illustrated below); and the *application* of a parameterised theory (such as **LIST**) to an actual theory (such as **NAT**) yielding a result theory (such as **LIST**[**NAT**]). Clear's parameterised theories also involve a *formal* or *requirement* theory **R** which makes explicit the syntactic *and* semantic requirements on an actual theory in order for the result of the application to be meaningful: **R** will be a subtheory of the body theory **B**. Moreover, a *binding* f of what the actual theory provides to what the formal theory requires is needed to carry out the application, called a "fitting morphism" in Clear. The semantics of Clear [15, 16] says that the *result* **P** of the application is given by the following pushout diagram, in which **P** is the resulting theory:



Similarly, the **sum T** of two theories **T1** and **T2** which share **NAT** and **BOOL** as subtheories, is given by the following colimit diagram, in which all lines indicate subtheory inclusions:

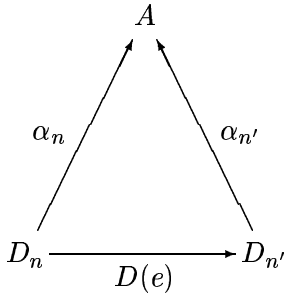


The construction of large theories as colimits of small theories connected by theory morphisms which was implicit in the first paper on Clear [15] was made explicit in [16] and [22]. For this approach to make sense, the category of theories should have finite colimits. For the equational institution, the intuitively correct syntactic pasting together of presentations exactly corresponds to colimits of theories, as proved in [45]. Colimits have become a

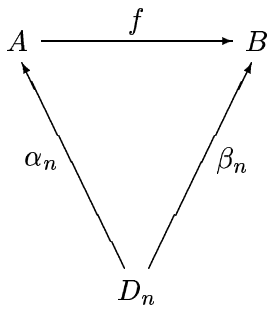
familiar technique in defining the semantics of specification languages, for example in ASL [89], Extended ML [87], Act One [25], and Act Two [29], as well as Clear and OBJ. Quite similar things can be done for logical programming languages, and even for conventional imperative languages like Ada [39, 104]. Thus, there is much motivation for studying colimits in the category of theories over an institution.

The use of colimits to explicate various ways to combine theories is not *ad hoc*. It is a general principle that a large widget can be described as the interconnection of a system of small widgets using widget-morphisms to identify the interfaces over which the interconnection is to be done; then the result of the interconnection is given by the colimit of the diagram; this principle may be found in [35, 36] from the point of view of general systems theory. Colimits have also been used for many other things in Computing Science [41], including graph grammars [23]. We now review the necessary categorical concepts.

Definition 9: A **diagram** D in a category $\mathbf{C}$ consists of a graph G together with a labelling of each node n of G by an object D_n of $\mathbf{C}$, and a labelling of each edge e, say from node n to node n' in G, by a morphism $D(e)$ in $\mathbf{C}$ from D_n to $D_{n'}$; let us write $D : G \to \mathbf{C}$. Then a **cone** α in $\mathbf{C}$ over the diagram D consists of an object A of $\mathbf{C}$ and a family of morphisms $\alpha_n : D_n \to A$, one for each node n in G, such that for each edge $e : n \to n'$ in G, the diagram



commutes in $\mathbf{C}$. We call D the **base** of the cone α, A its **apex**, G its **shape**, and we write $\alpha : D \Rightarrow A$. If α and β are cones with base D and apexes A, B (respectively), then a **morphism of cones** $\alpha \to \beta$ is a morphism $f : A \to B$ in $\mathbf{C}$ such that for each node n in G the diagram



commutes in **C**. Now let **Cone**$(D, \mathbf{C})$ denote the category of all cones over D in **C**, with the obvious composition. Then a **colimit** of D in **C** is an initial object in **Cone**$(D, \mathbf{C})$. □

The uniqueness of initial objects up to isomorphism implies the uniqueness of colimits up to cone isomorphism. The apex of a colimit cone α is called its **colimit object**, and the morphisms α_n to the apex are called the **injections** into the colimit. The colimit object is also unique up to isomorphism.

Definition 10: A category **C** is **finitely cocomplete** iff it has colimits of all finite diagrams, and is **cocomplete** iff it has colimits of all diagrams (whose base graphs are sets). A functor $F\colon \mathbf{C} \to \mathbf{C}'$ **reflects** [**creates**] **colimits** iff whenever D is a diagram in **C** such that the diagram $D;F$ in $\mathbf{C}'$ has a colimit cone $\alpha'\colon D \Rightarrow A'$ in $\mathbf{C}'$, then there is also a [unique] colimit cone $\alpha\colon D \Rightarrow A$ in **C** such that $\alpha' = \alpha;F$, i.e., such that $\alpha'_n = F(\alpha_n)$ for all nodes n in the base of D. □

Because colimits in the category of theories over an institution are to be used for putting together smaller specifications to form larger ones, it would be very nice to have a powerful, general criterion for when such colimits of specifications actually exist. Perhaps surprisingly, it suffices for the category of signatures to have colimits; the following more general result is the first really non-trivial result of this paper.

Theorem 11: The forgetful functor $Sign\colon \mathbf{Th} \to \mathbf{Sign}$ reflects colimits.

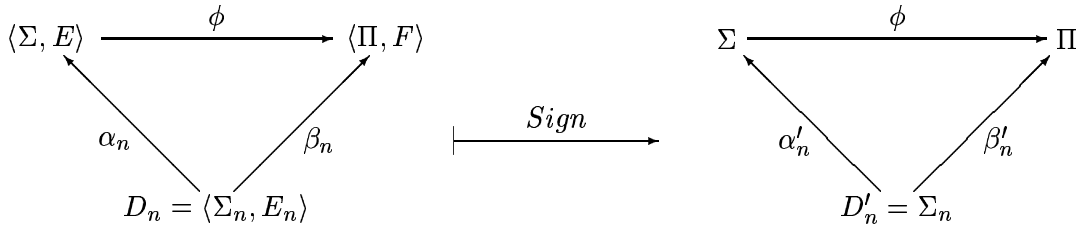
Proof: Suppose that $D\colon G \to \mathbf{Th}$ is a diagram in **Th**, say with $D_n = \langle \Sigma_n, E_n \rangle$ for $n \in |G|$. Let $D' = D;Sign\colon G \to \mathbf{Sign}$ be the corresponding diagram in **Sign**, in which $D'_n = \Sigma_n$. Now let $\alpha'\colon D' \Rightarrow \Sigma$ be a colimit cone for D'. We have to find a colimit cone α for D such that $Sign(\alpha) = \alpha'$. We define $D = \langle \Sigma, E \rangle$ where

$$E = (\bigcup_{n \in |G|} \alpha_n(E_n))^\bullet$$

and we define $\alpha_n = \alpha'_n$ for all $n \in |G|$. Then each α_n is a theory morphism and we claim that $\alpha = \langle \alpha_n \colon \langle \Sigma_n, E_n \rangle \to \langle \Sigma, E \rangle \mid n \in |G| \rangle$ is a colimit cone over D in **Th**. For suppose that $\beta'' = \langle \beta_n \colon \langle \Sigma_n, E_n \rangle \to \langle \Pi, F \rangle \mid n \in |G| \rangle$ is another cone over D in **Th**. Then applying *Sign* to everything, we get a unique $\phi \colon \Sigma \to \Pi$ such that $\alpha_n; \phi = \beta_n$ for each $n \in |G|$. Thus, there is at most one $\psi \colon \langle \Sigma, E \rangle \to \langle \Pi, F \rangle$ such that $\alpha_n; \psi = \beta_n$ for all $n \in |G|$, namely ϕ. Therefore all we need to show is that ϕ is a theory morphism. Because $\beta_n \colon \langle \Sigma_n, E_n \rangle \to \langle \Pi, F \rangle$ is a theory morphism, we have $\beta_n(E_n) \subseteq F$. Therefore $\bigcup_{n \in |G|} \beta_n(E_n) \subseteq F$, and so

$$\phi(E) = \phi((\bigcup_n \alpha_n(E_n))^\bullet) \subseteq (\bigcup_n \alpha_n; \phi(E_n))^\bullet = (\bigcup_n \beta_n(E_n))^\bullet \subseteq F$$

where the first inclusion follows from the Closure Lemma. (See the diagram below.) □



It now follows, for example, that the category **Th** of theories in an institution is finitely cocomplete if its category **Sign** of signatures is finitely cocomplete. (It is easy to see that the converse also holds.) Because the category of equational signatures is finitely cocomplete ([43, 46] give a simple proof using comma categories), we conclude that the category of signed equational theories is cocomplete. Using similar techniques, we can show that the category of first order signatures (as defined in Appendix A.2) is cocomplete, and thus without effort conclude that the category of first order theories is cocomplete (this might even be a new result, especially since our notion of morphism is not quite the usual one in logic).

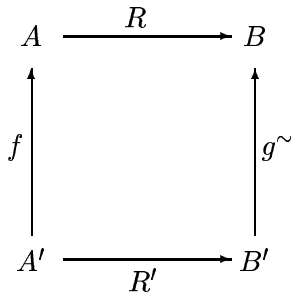
Actually, the stronger result holds, that *Sign creates* colimits.

2.5 A More Categorical Formulation

The formulation of institution given in Definition 1 leaves two important questions unanswered: (1) what about deduction? and (2) what are institution morphisms? We will see later on that institution morphisms are important for many applications and extensions of our basic theory, including the use of multiple institutions for specification, and the reusability of

theorem provers. Category theoretic intuition suggests that a more abstract formulation (than a family of relations) might help us with both of these questions. (The reader not already familiar with vertical and horizontal composition of natural transformations who wants to read this subsection might first consult [67], Section II.5.)

Let **Rel** denote the category with sets $A, B, C, \ldots$ as objects; with relations $R\colon A \to B$ as morphisms, i.e., triples $\langle A, R, B\rangle$ where $R \subseteq A \times B$; with the "diagonal" relation $\{\langle a, a\rangle \mid a \in A\}$ as the identity on a set A; and with composition as usual for relations, but keeping track of their sources and targets. Given a relation $R\colon A \to B$, let $R^{\sim}\colon B \to A$ denote its converse. We now define the category[3] **Trel** of "twisted relations": its objects are relations $R\colon A \to B$, and its morphisms from $(R\colon A \to B)$ to $(R'\colon A' \to B')$ are pairs of functions $\langle f\colon A' \to A,\ g\colon B \to B'\rangle$ such that the diagram



commutes in **Rel**, i.e., such that for all a' in A' and b in B, we have[4] $f(a')Rb$ iff $a'R'g(b)$. The identity morphism on a relation $R\colon A \to B$ is the pair $\langle 1_A, 1_B\rangle$, and the composition $\langle f', g'\rangle;\langle f, g\rangle$ is $\langle f'; f, g; g'\rangle$ provided the target sets of $\langle f', g'^{\sim}\rangle$ equal the source sets of $\langle f, g^{\sim}\rangle$.

The notation for a proper categorical definition of the institution concept will be simpler if we first define two functors on **Trel**. First, $Left\colon$ **Trel** $\to$ **Set**^{op} is defined by $Left(R\colon A \to B) = A$ and $Left(\langle f, g\rangle) = f$. Second, $Right\colon$ **Trel** $\to$ **Set** is defined by $Right(R\colon A \to B) = B$ and $Right(\langle f, g\rangle) = g$. Notice that $Left(f; f') = Left(f'); Left(f)$, whereas $Right(g; g') = Right(g); Right(g')$.

Definition 12: An **institution** is a functor $\mathcal{I}\colon$ **Sign** $\to$ **Trel** to twisted relations; its source **Sign** is its **category of signatures**; the functor composition $\mathcal{I}; Left\colon$ **Sign** $\to$ **Set**^{op} is the "model functor" of $\mathcal{I}$, denoted Mod; the composition $\mathcal{I}; Right\colon$ **Sign** $\to$ **Set** is the "sentence functor" of $\mathcal{I}$, denoted Sen; and the satisfaction relation $\models_{\Sigma}$ is $\mathcal{I}(\Sigma)$ for $\Sigma \in |\mathbf{Sign}|$. $\square$

[3] One referee wrote that Fred Linton introduced a similar (or identical) notion some time ago, but we have not found a precise reference.

[4] The reader may recognise the ghost of the Satisfaction Condition here.

This definition gives the version of institution in which each $Mod(\Sigma)$ is a set, rather than the more advanced version with a category $\mathbf{Mod}(\Sigma)$ of models. To get the category version, we use instead of **Trel** a category whose objects are triples $\langle \mathbf{A}, R, B \rangle$, where $\mathbf{A}$ is a category, B is a set, and $R \subseteq |\mathbf{A}| \times B$ is a relation; morphisms are then pairs $\langle F, g \rangle$ where $F : \mathbf{A}' \to \mathbf{A}$ is a functor and $g : B \to B'$ is a function, such that the above diagram commutes with $f = |F|$. Definition 12 with this category gives the desired concept. The following is now very natural:

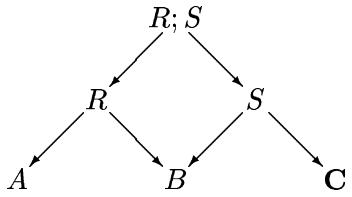
Definition 13: An **institution morphism** $\mathcal{I} \to \mathcal{I}'$ consists of a functor $\Phi : \mathbf{Sign} \to \mathbf{Sign}'$ and a natural transformation $\eta : \Phi; \mathcal{I}' \Rightarrow \mathcal{I}$. Given $\langle \Phi, \eta \rangle : \mathcal{I} \to \mathcal{I}'$ and $\langle \Phi', \eta' \rangle : \mathcal{I}' \to \mathcal{I}''$, their composition is defined to be $\langle \Phi; \Phi', (\Phi \circ \eta); \eta' \rangle : \mathcal{I}'' \to \mathcal{I}$, where $\circ$ denotes the (vertical) composition of a natural transformation with a functor. Let **INS** denote the category of institutions. □

There are reasons for believing that it might have been better to consider the above as defining a morphism from $\mathcal{I}'$ to $\mathcal{I}$, rather than from $\mathcal{I}$ to $\mathcal{I}'$, even though in a certain sense the choice is arbitrary; however, we have chosen to maintain the above definition in this paper for essentially historical reasons. Section 4.1 gives a more concrete version of this definition. That **INS** really is a category, and moreover has whatever limits and colimits **Trel** has, follows from some "abstract nonsense" and "general systems theory"[5] described in the following subsection.

2.5.1 An Even More Categorical Formulation

In order to consider variations of the notion of institution, and also to study properties of the category of institutions, it is helpful to take a very general approach. Readers who are not especially fond of categorical "abstract nonsense" may wish to skip this subsection; also, it is somewhat sketchy. Given a category $\mathbf{C}$ with pullbacks, we first define the category $\mathbf{Rel}(\mathbf{C})$ of **relations** in $\mathbf{C}$ as follows: its objects are those of $\mathbf{C}$; its morphisms from A to B are pairs $\langle p1 : R \to A, p2 : R \to B \rangle$ of morphisms in $\mathbf{C}$ with a common source; its identities have both $p1$ and $p2$ identity morphisms; and its **composition** is obtained by pullback, as shown in the diagram below:

[5] An earlier draft of this definition had the natural transformation going the other way; comparison with the general systems theory in [35, 36] enabled us to formulate institution morphisms correctly.



Lemma 14: If $\mathbf{C}$ has pullbacks, then $\mathbf{Rel}(\mathbf{C})$ is a category. □

We next recall Lawvere's general **comma category** construction. If we are given functors $F\colon\ \mathbf{A} \to \mathbf{C}$ and $G\colon\ \mathbf{B} \to \mathbf{C}$, then the category (F/G) is defined as follows: its objects are triples $\langle A, c\colon\ F(A) \to G(B), B\rangle$, where A is an object of $\mathbf{A}$, B is an object of $\mathbf{B}$, and c is a morphism of $\mathbf{C}$; its morphisms from $\langle A, c\colon\ F(A) \to G(B), B\rangle$ to $\langle A', c'\colon\ F(A') \to G(B'), B'\rangle$ are pairs $\langle a,\ b\rangle$ with $a\colon\ A \to A'$ in $\mathbf{A}$ and $b\colon\ B \to B'$ in $\mathbf{B}$ such that $F(a); c' = c; G(b)$; and the composition $\langle a,\ b\rangle; \langle a',\ b'\rangle$ in (F/G) is just $\langle a; a',\ b; b'\rangle$. It is known[6] that this gives a category such that whatever limits or colimits are possessed by $\mathbf{A}$ and $\mathbf{B}$ and preserved by F and G, are also possessed by (F/G).

Next we give some ways to construct functors into relation categories. Given $F\colon\ \mathbf{A} \to \mathbf{C}$, define $F{\uparrow}\colon\ \mathbf{A} \to \mathbf{Rel}(\mathbf{C})$ as follows: an object A in $\mathbf{A}$ goes to the object $F(A)$ in $\mathbf{Rel}(\mathbf{C})$; and a morphism $a\colon\ A \to A'$ in $\mathbf{A}$ goes to a morphism $\langle 1_{F(A)},\ F(a)\rangle$ from $F(A)$ to $F(A')$ in $\mathbf{Rel}(\mathbf{A})$. (It is easy to see that this is a functor.) Similarly, given $F\colon\ \mathbf{A} \to \mathbf{C}$, define $F{\downarrow}\colon\ \mathbf{A}^{op} \to \mathbf{Rel}(\mathbf{C})$ by: an object A in $\mathbf{A}$ goes to the object $F(A)$ in $\mathbf{Rel}(\mathbf{C})$; and a morphism $a\colon\ A \to A'$ in $\mathbf{A}^{op}$ goes to a morphism $\langle 1_{F(A')},\ F(a)\rangle$ from $F(A')$ to $F(A)$ in $\mathbf{Rel}(\mathbf{A})$. (It is also easy to see that this is a functor.) This last construction will let us "twist" relations.

Now let's exercise this machinery. Combining the relation, comma category, and the $\uparrow$ and $\downarrow$ constructions will give us the categories of twisted relations that we need; and taking functors into these will give us the corresponding categories of institutions. Let's get the relation categories first. The (original) definition of institution with *Mod* set-valued corresponds to the twisted relation category $(1_{\mathbf{Set}}{\downarrow}\ /\ 1_{\mathbf{Set}}{\uparrow})$ where $1_{\mathbf{A}}$ denotes the identity functor on the category $\mathbf{A}$. Similarly, Definition 1 corresponds to the category $(1_{\mathbf{Set}}{\downarrow}\ /\ U{\uparrow})$, where $U\colon\ \mathbf{Cat} \to \mathbf{Set}$ is the forgetful functor that takes each category to its underlying set (or class), and each functor to its underlying function on objects. Both enjoy the "twist" given by $\downarrow$, but the first gives only sets of models, while the second gives categories of models; in each case, the source category of the first functor gives the structure for models.

[6] Actually, somewhat stronger results are known; see [46, 100].

A natural and interesting variant is given by the category $(U\downarrow / U\uparrow)$, which allows morphisms between sentences as well as between models; one might want to think of a morphism from one sentence to another as a "proof" that the second follows from the first; see [47] and Section 5 for further discussion of this point.

An institution is a functor into a relation category; in particular, an institution in the sense of Definition 1 is a functor $\mathcal{I} : \mathbf{Sign} \to (1_{\mathbf{Set}}\downarrow / U\uparrow)^{op}$. This leads to categories whose objects are functors; but because their source categories (which are their categories of signatures) may vary, their morphisms will not be just natural transformations. In fact, these morphisms are pairs of the form $\langle \Phi : \mathbf{Sign} \to \mathbf{Sign}', \eta : \Phi; \mathcal{I}' \Rightarrow \mathcal{I} \rangle$, and are a special case of a general construction for the category of diagrams (with varying shape) over a category: Given a category $\mathbf{T}$, the objects of $\mathbf{Dgm}(\mathbf{T})$ are functors $\mathcal{I} : \mathbf{S} \to \mathbf{T}$, its morphisms $\mathcal{I} \to \mathcal{I}'$ are pairs $\langle \Phi : \mathbf{S} \to \mathbf{S}', \eta : \Phi; \mathcal{I}' \Rightarrow \mathcal{I} \rangle$, and the composition $\langle \Phi : \mathbf{S} \to \mathbf{S}', \eta : \Phi; \mathcal{I}' \Rightarrow \mathcal{I} \rangle; \langle \Phi' : \mathbf{S}' \to \mathbf{S}'', \eta' : \Phi; \mathcal{I}'' \Rightarrow \mathcal{I}' \rangle$ is $\langle \Phi; \Phi', \Phi \circ (\eta'; \eta) \rangle$, where in the last expression "$\circ$" denotes the horizontal and ";" denotes the vertical composition of natural transformations. That $\mathbf{Dgm}(\mathbf{T})$ is a category is shown, for example, in [36] in the context of general systems.

This abstract view is also useful for getting other variants of the institution notion; for example, to get partial satisfaction we might let $\mathbf{C}$ be the category $\mathbf{Pfn}$ of partial functions; it seems worth exploring this further. This view of institutions also allows us to get completeness results for the category of institutions using general results about relation, diagram, and comma categories; see also [100]. [47] discusses an even more abstract formulation based on "wedges."

3 Constraints

To avoid over specifying problems, we sometimes want to use *loose* specifications, that is, specifications for which non-isomorphic models are acceptable. On the other hand, most problems require some fixed data types, such as the natural numbers or truth values. In such cases, the subtheories that correspond to these data types must be given *standard* interpretations. Finally, sometimes we want to consider *parameterised* standard data types, such as $\mathbf{SET}[\mathbf{X}]$ and $\mathbf{LIST}[\mathbf{X}]$, for which sets and lists are to be given standard interpretations, once an interpretation is given for $\mathbf{X}$, which is loose with respect to some requirement theory.

We have already considered the category $T^* = \mathbf{Mod}(T)$ of *all* (loose) interpretations of a theory T over a fixed institution; now we consider how to impose constraints on these interpretations. One kind of constraint requires

that some parts of T have a "standard" interpretation relative to other parts; these are the "data constraints" of [17], which generalise and relativise the "initial algebra" approach to abstract data types of ADJ [54], and slightly generalize the "canons" of Reichel [84]. Data constraints make sense for any institution, and are much more expressive than just initiality, even for the equational institution.

To require that some subtheories $T_1, \ldots, T_n$ of a theory T are initially interpreted, we could try to use the subtheories themselves as constraints. In this case, a model M would satisfy T with constraint $(T_1, \ldots, T_n)$ iff M satisfies T, and M restricted to each T_i is initial. More precisely now, given $F_i : T_i \to T$ for $i = 1, \ldots, n$, we can define a satisfaction relation $\models$ between models and theories with constraints by

$$M \models \langle T, (F_1, \ldots, F_n) \rangle \text{ iff } M \models T \text{ and } \mathbf{Mod}(F_i)M \text{ is initial in } \mathbf{Mod}(T_i) \text{ for } i = 1, ..., n.$$

Unfortunately, the above definition is too naive. First, to deal with parameterised theories, we need not just initial models (e.g., for the natural numbers), but we also need free extensions of models (e.g., to form sets from elements). Second, it would be very convenient if for any institution $\mathcal{I}$, we could construct a new institution $\mathcal{C}(\mathcal{I})$ with sentences either $\mathcal{I}$-sentences or else constraints, and with its models the $\mathcal{I}$-models. For this to work, we need translations of constraints under signature morphisms. Then, if $\mathcal{I}$-signatures have colimits, Theorem 11 will imply that $\mathcal{C}(\mathcal{I})$-theories also have colimits, and thus can be glued together. Unfortunately, there is no obvious way to translate the naive constraints; however, Definition 19 below gives a notion that is sufficiently general and also admits translations.

Section 3.3 will generalise constraints further, to include so-called "generating constraints" and Section 4.2 will generalise to "duplex constraints" that allow constraints in an institution different from the one in which models are taken; more generally still, Section 4.3 will consider multiplex institutions.

3.1 Free Interpretations

Suppose that we want to define the natural numbers in the equational institution. To this end, consider a theory $\mathbf{N}$ with one sort **Nat**, and with a signature Σ containing one constant 0 and one unary operator *inc*; there are no equations in the presentation of this theory. Now this theory has many algebras, including some where $inc(0) = 0$. But the natural numbers, no matter how represented, give an **initial** algebra in the category $\mathbf{Alg}_\Sigma$ of all algebras for this theory, in the sense that it has exactly one Σ-homomorphism to any other Σ-algebra. It is easy to prove that any two initial Σ-algebras are Σ-isomorphic; this means that the property "being initial" determines the

natural numbers uniquely up to isomorphism, that is, different representations for the natural numbers give different (but isomorphic) initial algebras. This characterisation of the natural numbers is due to Lawvere [70], and a proof that it is equivalent to Peano's axioms can be found in [68], pages 67-70.

The initial algebra approach to abstract data types [54] takes this "Lawvere-Peano" characterisation of the natural numbers as paradigmatic for defining other abstract data types; the method has been used to specify sets, lists, stacks, and many many other data types, as well as database systems and programming languages, among many other things. The essential ideas here are that concrete data types are algebras, and that "abstract" in "abstract data type" means *exactly* the same thing as "abstract" in "abstract algebra," namely, uniquely defined up to isomorphism. A number of less abstract equivalents to initiality for the equational institution, including generalised Peano axioms, are given in [79].

Let us now consider the case of the parameterised abstract data type of sets of elements of a sort **S**. We add a new sort **Set**, and operators[7]

$$\emptyset : \rightarrow \mathbf{Set}$$
$$\{\_\} : \mathbf{S} \rightarrow \mathbf{Set}$$
$$\_ \cup \_ : \mathbf{Set}, \mathbf{Set} \rightarrow \mathbf{Set}$$

subject to the following equations, where U, U' and U'' are variables of sort **Set**,

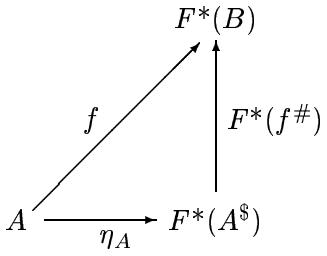
$$\emptyset \cup U = U$$
$$U \cup (U' \cup U'') = (U \cup U') \cup U''$$
$$U \cup U' = U' \cup U$$
$$U \cup U = U$$

Although we want these operators to be interpreted "initially" in some sense, we do *not* want the initial algebra of the theory having sorts **S** and **Set** and the operators above. Indeed, the initial algebra of this theory has the *empty* carrier for the sort **S** (because there are no operators to generate elements of sort **S**) and has only the element $\emptyset$ of sort **Set**. Rather, we want to permit *any* interpretation for the parameter sort **S**, and then require that the new sort **Set** and its new operators are interpreted initially *relative* to the given interpretation of **S**.

Let us make this precise. Suppose that $F\colon T \rightarrow T'$ is a theory morphism. Then there is a forgetful functor from the category of T'-models to the category of T-models, $F^*\colon T'^* \rightarrow T^*$ as in Definition 6. For the equational case, a very general result of Lawvere [69] says that for every T-model A

[7] We use "mixfix" declarations in this signature, in the style of OBJ: each underbar is a placeholder for an element of the corresponding sort from the sort list between the colon and the arrow.

there is a T'-model $A^\$$ and a T-homomorphism $\eta_A \colon A \to F^*(A^\$)$, called the **universal morphism** and characterised by the following "universal property": given any T'-model B and any T-morphism $f \colon A \to F^*(B)$, there is a unique T'-morphism $f^\# \colon A^\$ \to B$ such that the following diagram commutes in $\mathbf{Mod}(T)$:



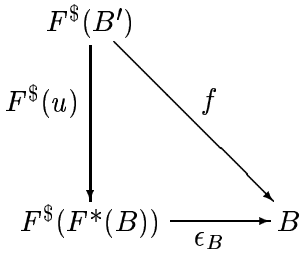
Such an object $A^\$$ in $\mathbf{Mod}(T')$ will be called a **free extension of A along F**; the universal property determines $A^\$$ uniquely up to isomorphism in $\mathbf{Mod}(T')$. This discussion motivates the following:

Definition 15: A theory morphism $F \colon T \to T'$ is **liberal** iff for every T-model A, there is a T'-model $A^\$$, called the **free extension of A along F**, such that there is a **universal** morphism $\eta_A \colon A \to F^*(A^\$)$ with the property that for each T'-model B and each T-morphism $f \colon A \to F^*(B)$, there is a unique T'-morphism $f^\# \colon A^\$ \to B$ such that $\eta_A ; F^*(f^\#) = f$ (in the category $\mathbf{Mod}(T)$). An institution is **liberal** iff each of its theory morphisms is liberal. □

In any institution, if we are given a free extension $A^\$$ along F for each A in $\mathbf{Mod}(T)$, then there is unique way to define a functor $F^\$ \colon \mathbf{Mod}(T) \to \mathbf{Mod}(T')$ such that $F^\$(A) = A^\$$; then $F^\$ \colon T^* \to T'^*$ is called the **free (extension) functor** determined by F. (This follows from a general result about adjoints [67], noting that $F^\$$ is a left adjoint to F^*, and η is a natural transformation, the unit of this adjunction. In particular, an institution is liberal iff the forgetful functors induced by its theory morphisms always have left adjoints.) The equational institution is liberal, as are the institutions of Horn clause logic with equality, and of conditional equations; however the first order logic institution is not liberal (the latter three institutions are defined in Appendix A.2). Notice that even in a non-liberal institution, there may be many models that have a free extension along a given theory morphism, and there may also be many theory morphisms that have a free extension functor. Hence, in the following we may use the notation $F^\$(A)$ for a free extension of A along F even when there is no free functor $F^\$$ for F.

Returning to our set example, consider the theory morphism Set that is the inclusion of the trivial theory **TRIV** having just the sort **S**, into the theory of sets of **S**, let's call it **SET**, obtained by adding the sort **Set** and the operators and equations given above. Then Set^* takes a **SET**-algebra and forgets the new sort **Set** and the three new operators, giving an algebra that has just the carrier of **S**-sorted elements. The free functor $Set^\$$ takes a **TRIV**-algebra (i.e., a set) A and extends it freely to a **SET**-algebra, with the new operators giving distinct results except where the equations of **SET** force equality. This ensures that $Set^\$(A)$ is the algebra of all finite subsets of A.

Given a **SET**-algebra B, there is a natural way to check whether or not its **Set** sort and operators are free over its parameter sort **S**: let $A = Set^*(B)$ in the above diagram, and let $f = 1_{Set^*(B)}$; then $f^\#\colon (Set^*(B))^\$ \to B$ should be an isomorphism. In general, the morphism $(1_{F^*(B)})^\#\colon (F^*(B))^\$ \to B$ is called the **counit** (of the adjunction, if there is one) and is denoted ϵ_B. In a liberal institution, it has the following **couniversal** property, dual to that of the unit: given any B' in $\mathbf{Mod}(T')$ and $f\colon F^\$(B') \to B$ in $\mathbf{Mod}(T)$, there is a unique $u\colon B' \to F^*(B)$ such that $F^\$(u); \epsilon_B = f$, i.e., such that the diagram



commutes in $\mathbf{Mod}(T)$. This motivates the following.

Definition 16: Let $F\colon T \to T'$ be a theory morphism. Then a T'-model B is F-**free** iff $F^*(B)$ has a free extension $F^\$(F^*(B))$ along F such that $\epsilon_B = (1_{F^*(B)})^\#$ is an isomorphism. Let us call a functor $G\colon \mathbf{Mod}(T) \to \mathbf{Mod}(T')$ **extensive** iff $G(A)$ is F-free for every T-model A. □

Of course, we are mostly interested in whether or not a free extension functor is extensive. The notion of F-free for the equational case is due to Thatcher, Wagner and Wright [103]; here, we generalise to arbitrary institutions, without assuming that there is necessarily a free functor $F^\$$. Originally, we defined B to be F-free if B and $F^\$(F^*(B))$ were isomorphic [17]. However, there are examples[8] where these two objects are isomorphic, but

[8] Our thanks to Eric Wagner for this comment.

not naturally so by the counit morphism ϵ_B. There is also a concept that is dual to F-free in a certain sense:

Definition 17: Let $F: T \to T'$ be a theory morphism. Then a T-model A is F-**protected** iff it has a free extension $A^\$$ along F such that the universal morphism $\eta_A: A \to F^*(A^\$)$ is an isomorphism. Moreover, the free functor $F^\$$ is **persistent** iff every T-model A is F-protected, using the free extension $F^\$(A)$. □

This definition generalises the concept of persistence in [103] to any institution. Let us define a theory morphism $F: T \to T'$ to be **conservative** iff $e \notin T$ implies $F(e) \notin T'$. A syntactic characterisation of persistence for the equational institution is given by Goguen and Meseguer in [49][9]: when F is injective on sorts, $F^\$$ is persistent iff it is conservative and whenever t' is a Σ'-term with its variables and its sort in $F(T)$, then there is an equation $(\forall X)\, t' = F(t)$ in T' with t a Σ-term (where Σ is the signature of T and Σ' is the signature of T'). Unfortunately, this result uses some rather subtle constructions in the equational institution which it is not obvious how to generalise to other institutions. The following simple result relates the concepts of Definitions 16 and 17.

Proposition 18: Let $F: T \to T'$ be a theory morphism in a liberal institution.

1. If a T-model A is F-protected then $F^\$(A)$ is F-free.
2. If a T'-model A' is F-free then $F^*(A')$ is F-protected.
3. If $F^\$$ is persistent then $F^\$$ is extensive.

Proof: For the first assertion, we use the unit isomorphism $\eta_A: A \to F^*(F^\$(A))$ and the fundamental equation for the adjunction (see [67], Theorem IV.2, p.81), $F^\$(\eta_A); \epsilon_{F^\$(A)} = 1_{F^\$(A)}$, to see that

$$\epsilon_{F^\$(A)}: F^\$(F^*(F^\$(A))) \to F^\$(A)$$

is also an isomorphism. The second assertion is proved in the same way. The third assertion, that $F^\$$ persistent implies $F^\$(A)$ is F-free for every A, follows directly from the first. □

[9] The result in [49] is slightly more general, because it allows theory morphisms that map operators to terms.

The following remarks may be of some interest to those familiar with adjoint functors: Let $F: T \to T'$ be a liberal theory morphism. Then $F^\$$ is persistent iff the unit $\eta: 1_{\mathbf{Mod}(T)} \Rightarrow F^\$; F^*$ of the adjunction between $F^\$$ and F^* is an isomorphism. Moreover, in this case $F^\$$ can be chosen so that this isomorphism is actually an equality, i.e., such that $F^\$$ is what [103] call **strongly persistent**.

Results of Mahr and Makowsky [75] show that there is a sense in which the most general sublanguage of infinitary first order logic admitting initial models is Horn clause logic with infinitary conditions; further, the most general finitary sublanguage uses finitary Horn clauses; the most general equational sublanguage uses (infinitary) conditional equations; and the most general finitary equational sublanguage consists of finitary conditional equations. These results are in the style of abstract model theory, a framework much closer to first order logic than to institutions; also, the results concern the existence of initial models rather than left adjoints to forgetful functors. Tarlecki has extended this work of Mahr and Makowsky to characterise liberal sublanguages [99], and has also generalised to what he calls "abstract algebraic institutions" [98].

3.2 Constraining Theories

It is very convenient in program specification to distinguish between sets of sentences which are to be interpreted "loosely" (i.e., any model satisfying the theory will do) and those which require a standard interpretation, say an initial model, or more generally, a free extension. For example, in a parameterised specification, we may want to permit any interpretation of the parameter (say, any partially ordered set) but also want to constrain the interpretation of the specification which enriches this parameter to be free over it (say, strings of elements of the set, instead of just some arbitrary monoid over them).

Thus, in a specification language, we may want to let theories include not just the sentences provided by some institution, but also sentences that constrain certain subtheories to be interpreted freely relative to others, in the sense of Section 3.1. We call such sentences **constraints** and we call theories that can include them **constraining theories**. For example, in a first order theory we might introduce a sentence which constrains interpretations of a theory of the natural numbers to be standard, i.e., to actually be the natural numbers. Our approach will provide a general logic-indepenedent way of moving to a more powerful language that can impose such constraints. We will show that theories with such constraints in them can be treated just like ordinary theories. Moreover, we will show that an institution whose

signatures can be glued together (with colimits) yields another institution whose theories may have constraints, such that these theories can also be glued together (with colimits).

Returning to the example of the previous subsection, let us further enrich **SET** with some new sorts and operators to get a theory **SETCH** which adds an operator

$$\mathit{choose} : \mathbf{Set} \to \mathbf{S}$$

that chooses an arbitrary element of any non-empty set, as described by the conditional equation

$$\mathit{choose}(U) \in U \quad \text{if} \quad U \neq \emptyset$$

where U is a variable of sort **Set**. This is a good example of a loose specification. Now let $\mathit{Etc} : \mathbf{SET} \to \mathbf{SETCH}$ be the theory inclusion, with $\mathit{Etc} : \mathbf{Sign}(\mathbf{SET}) \to \mathbf{Sign}(\mathbf{SETCH})$ the corresponding signature morphism. Then a **SETCH**-algebra A interprets sets as intended iff $\mathit{Etc}^*(A)$ satisfies **SET** and is *Set*-free. This motivates

Definition 19: Let Σ be a signature. Then a Σ-**constraint** is a pair

$$\langle F : T'' \to T',\ \theta : \mathit{Sign}(T') \to \Sigma \rangle$$

consisting of a theory morphism and a signature morphism. (We may call a Σ-constraint a Σ-**data constraint** if it is used to define a data type.) A Σ-model A **satisfies** the Σ-constraint $c = \langle F : T'' \to T',\ \theta : \mathit{Sign}(T') \to \Sigma \rangle$ iff θA satisfies T' and is F-free, which means that $F^*(\theta A)$ has a free extension along F such that $(1_{F^*(\theta A)})^{\#} = \epsilon_{\theta A} : F^{\$}(F^*(\theta A)) \to \theta(A)$ is an isomorphism; in this case we write $A \models_\Sigma c$. $\square$

Notice that T'' may be the empty theory. The following picture of the situation in this definition may help:

$$\begin{array}{ccc} T'' & \xrightarrow{F} & T' \\ \mathit{Sign}(T') & \xrightarrow{\theta} & \Sigma \end{array}$$

$$\begin{array}{ccc} T''* & \underset{F^{\$}}{\overset{F^*}{\rightleftarrows}} & T'* \\ \mathbf{Mod}(\Sigma') & \xleftarrow{\theta^*} & \mathbf{Mod}(\Sigma) \end{array}$$

In our set example, $F : T'' \to T'$ is the inclusion $\mathit{Set} : \mathbf{TRIV} \to \mathbf{SET}$ and $\theta : \mathit{Sign}(T') \to \Sigma$ is the signature morphism that underlies the theory inclusion $\mathit{Etc} : \mathbf{SET} \to \mathbf{SETCH}$. For any Σ-algebra A, it makes sense

to ask whether θA satisfies T' and is F-free, as in Definition 19. It is not necessary that the institution involved is liberal.

Our work on constraints dates from the Spring of 1979, and was influenced by a lecture of Reichel in Poland in 1978 and by the use of functors to handle parametric data types by Thatcher, Wagner and Wright in [103]. Historically the first work in this area seems to be the little known 1971 paper [65] of Kaphengst and Reichel, which apparently considered the case of a single chain of theory inclusions. This was later generalised to "initially restricting algebraic theories" by Reichel [84]; see also [85]. There are three main differences between our "data constraints" and the "initial restrictions" of [84]: first, an initial restriction on an algebraic theory is a pair of subtheories, whereas we use a pair of theories with an arbitrary theory morphism between them, plus a signature morphism from the target theory. It seems very natural to use subtheories, but this does not give rise to an institution[10]; also, the added generality (of arbitrary theory morphisms) seems to permit some interesting additional examples. The second difference is simply that we are doing our work over an arbitrary institution. The third difference lies in the manner of adding constraints: whereas [84] defines a "canon" to be an algebraic theory together with some initial restrictions on it, we will define a new institution whose sentences on a signature Σ include both the old Σ-sentences and also Σ-constraints. This route has the technical advantage that both kinds of new sentence refer to the same signature. Moreover, we can hope to apply general results about institutions to these new institutions. The constraints of Ehrig, Wagner and Thatcher [28] add a third component $\theta_0 : \Sigma_0 \to \Sigma$ to the two in our notion, in order to deal with "derive;" their paper treats the equational institution, and requires that the model functor **Mod** preserve pushouts.

We now show that constraints behave like sentences even though they have a very different structure. Like sentences, they impose restrictions on the allowable models. Moreover, a signature morphism from Σ to Σ' translates Σ-constraints to Σ'-constraints, just as it translates from Σ-sentences to Σ'-sentences.

Definition 20: Let $\phi : \Sigma \to \Sigma'$ be a signature morphism and let $c = \langle F, \theta \rangle$ be a Σ-constraint. Then the **translation** of c by ϕ is the Σ'-constraint $\langle F, \theta; \phi \rangle$; we write this as $\phi(c)$. □

It is the need for the translation of a constraint to be a constraint that leads to constraints in which θ is not an inclusion. We now prove the Satisfaction Condition for constraints.

[10] Andrzej Tarlecki has shown us a counterexample.

Lemma 21: Constraint Satisfaction. If $\phi\colon \Sigma \to \Sigma'$ is a signature morphism, if c is a Σ-constraint, and if B is a Σ'-model then

$$B \models \phi(c) \text{ iff } \phi(B) \models c.$$

Proof: Suppose that $c = \langle F, \theta \rangle$ with $F\colon T'' \to T'$ and $\theta\colon Sign(T') \to \Sigma$. Then $B \models \phi(c)$ means that $B \models \langle F, \theta; \phi \rangle$, which means that $(\theta; \phi)(B)$ satisfies T' and is F-free. On the other hand, $\phi(B) \models c$ means that $\phi(\theta(B))$ satisfies T' and is F-free. To show that these are the same, we must show that $(\theta; \phi)(B) = \phi(\theta(B))$; but this is just the functoriality of **Mod**. □

Given an institution $\mathcal{I}$, we can construct another institution having as its sentences both the sentences of $\mathcal{I}$ and also constraints. This construction is of particular interest for liberal institutions, where we know that there is always a counit morphism with the couniversal property.

Definition 22: Given an arbitrary institution $\mathcal{I}$, construct the institution $\mathcal{C}(\mathcal{I})$ whose theories contain both constraints and $\mathcal{I}$-sentences as follows: the category of signatures of $\mathcal{C}(\mathcal{I})$ is the category **Sign** of signatures of $\mathcal{I}$; if Σ is an $\mathcal{I}$-signature, then $Sen_{\mathcal{C}(\mathcal{I})}(\Sigma)$ is the (disjoint) union of the collection $Sen_{\mathcal{I}(\Sigma)}$ of all Σ-sentences from $\mathcal{I}$ with the collection of all Σ-constraints[11]; also $\mathbf{Mod}(\Sigma)$ is the same for $\mathcal{C}(\mathcal{I})$ as for $\mathcal{I}$; we use the constraint translation of Definition 20 to define $Sen(\phi)$ on constraints; finally, satisfaction for $\mathcal{C}(\mathcal{I})$ is as in $\mathcal{I}$ for Σ-sentences from $\mathcal{I}$, and is as in Definition 19 for Σ-constraints. □

Proposition 23: If $\mathcal{I}$ is an institution, then $\mathcal{C}(\mathcal{I})$ is an institution.

Proof: Clearly the first condition of Definition 1 is satisfied, because the signature category of $\mathcal{C}(\mathcal{I})$ is that of $\mathcal{I}$. For the second condition we must show the functoriality of $Sen_{C(\mathcal{I})}$. Let $\phi\colon \Sigma \to \Sigma'$ in **Sign**; then $Sen_{\mathcal{C}(\mathcal{I})}(\phi)\colon Sen_{\mathcal{C}(\mathcal{I})}(\Sigma) \to Sen_{\mathcal{C}(\mathcal{I})}(\Sigma')$ is the disjoint union of the map $Sen_{\mathcal{I}}(\theta)$ defined on $Sen_{\mathcal{I}}(\Sigma)$ and the map sending Σ-constraints to their translations by ϕ. Because $Sen_{\mathcal{I}}$ is already functorial, it suffices to prove functoriality of constraint translation. But this is obvious.

There is nothing to check for the third condition. This leaves the Satisfaction Condition: we already know that it is satisfied for $\mathcal{C}(\mathcal{I})$-sentences that are $\mathcal{I}$-sentences; and Lemma 21 shows that it is satisfied for constraints. □

[11] There are some foundational questions about the size of the closure of a constraint theory that we will ignore here; they can be solved for example by limiting the size of the category of signatures used in the original institution.

This means that all the concepts and results of Section 2 can be applied to theories that include constraints as well as sentences, that is, to $\mathcal{C}(\mathcal{I})$-theories[12]. We call such theories **constraining theories**. Thus, we get notions of presentation and closure, as well as theory. In particular, the Presentation and Closure Lemmas hold, and we also get the following important result, which enables us to glue together constraining theories:

Theorem 24: Given an institution with a [finitely] cocomplete category of signatures, then its category of constraining theories is also [finitely] cocomplete.

Proof: Immediate from Theorems 11 and 23. □

Let us consider what this means for the equational institution. While rules of deduction for inferring that an equation is in the closure of a set of equations are familiar, we have no such general rules for constraints. However, it should be possible to obtain such rules (although they will not be first order, and in general will not be complete [73]) because a constraint corresponds to an induction principle plus some inequalities [18, 79]; in particular, the constraint that sets are to be interpreted freely gives us all the consequences of structural induction (in the classical sense [13]) for sets. In more detail, this constraint for sets demands that all elements of sort **Set** are generated by the operators $\emptyset, \{\_\}$ and $\_ \cup \_$, which can be expressed as a principle of structural induction over these generators (but note that structural induction is not first order). The constraint also demands that two elements of sort **Set** are *unequal* unless they can be proved equal using the given equations. One way to express this distinctness is to add a new boolean-valued binary operator on the new sort, say $\equiv$, with some new equations such that $t \equiv t' = \mathit{false}$ iff $t \neq t'$, and also such that $\mathit{true} = \mathit{false}$ does not hold [79]. Or we might consider an institution with disequations (see [14] for some related discussion).

Let us now consider an easier example, involving equational theories with the following three (unconstrained) presentations:

1. **E** – the empty theory: no sorts, no operators, no equations.
2. **N** – the theory with one sort **Nat**, one constant 0, one unary operator *inc*, and no equations.
3. **NP** – the theory with one sort **Nat**, two constants 0 and 1, one unary operator *inc*, one binary infix operator $+$, and equations $0 + n = n$, $\mathit{inc}(m) + n = \mathit{inc}(m + n)$, $1 = \mathit{inc}(0)$.

[12]Note that there is no reason to suppose that $\mathcal{C}(\mathcal{I})$ is liberal if $\mathcal{I}$ is.

Let Σ^N and Σ^{NP} be the signatures of **N** and **NP** respectively, and let F^N: **E** $\rightarrow$ **N** and F^{NP}: **N** $\rightarrow$ **NP** denote the inclusion morphisms. Now **NP** as it stands has many different interpretations, for example the integers modulo 10, or the truth values with $0 = \mathit{false}$, $1 = \mathit{true}$, $\mathit{inc}(\mathit{false}) = \mathit{false}$, $\mathit{inc}(\mathit{true}) = \mathit{true}$, $\mathit{false} + n = \mathit{false}$, $\mathit{true} + n = \mathit{true}$. In the latter model, $+$ is not commutative. In order to get the *standard* model suggested by the notation, we need to impose the constraint $\langle F^N, \mathit{Sign}(F^{NP})\rangle$ on the theory **NP**. Then the only model (up to isomorphism) is the natural numbers with the usual addition. Note that the equation $m+n = n+m$ is satisfied by this model and therefore appears in the equational closure of the presentation; it is a property of $+$ provable by induction. There are also extra constraints in the closure, for example, $c' = \langle F', \phi\rangle$, where F' is the inclusion of the empty theory **E** into the theory with 0, 1 and $+$ with identity, associativity and commutativity equations, and ϕ is its inclusion into **NP**. This constraint is satisfied in all models that satisfy the constraint $\langle F^N, \mathit{Sign}(F^{NP})\rangle$. In this sense, c' is a derived induction principle. Further examples can be found in [18].

In general, the closure of a constraining presentation to a constraining theory adds some new equations derivable by induction principles corresponding to the constraints; and it also adds some new constraints corresponding to derived induction principles. The new equations are important in giving a precise semantics for programming methodology. For example, we may want to supply **NP** as an actual parameter theory to a parameterised theory whose requirement demands a commutative binary operator. The new constraints seem less essential. For the Horn clause institution, constraints enable us to define predicates by induction, as discussed further at the end of Section 4.

What may be a promising approach to the proof-theoretic aspect of constraining theories is discussed by Clark [20], Reiter [86], McCarthy [77], and others who have been concerned with ways to get new sentences that are satisfied under a "closed world" assumption (the common sense assumption that the information actually given about a predicate is all the relevant information about that predicate; McCarthy identifies this with Occam's famous razor). We, of course, identify that "closed world" with the initial model of the given theory. Clark's scheme, called "predicate completion," is simply to infer the converse of any Horn clause. This is sound for ground queries in the institutions of conditional equations and first order Horn clauses (in the sense that all the answers thus obtained are true of the initial model); but it is not complete [94]. McCarthy calls his scheme "circumscription" and is interested in its application in the context of full first order logic [77]; it has been shown sound when minimal models exist, but can be unsound when

such models do not exist.

It is worth considering what happens if we add extra silly equations to **NP** constrained by $\langle F^N, F^{NP}\rangle$. For example, $1 + n = n$ contradicts the constraint. In fact, if we add it, we simply get a constraining theory which is *inconsistent*, in the sense that it has no models.

There is a rather elegant construction that gives a somewhat different notion of constraining theory than that given by $\mathcal{C}(\mathcal{I})$. Let $\mathcal{I}$ be an institution. Then the signature category of $\mathcal{C}'(\mathcal{I})$ is the category of theories of $\mathcal{I}$. If T is a theory of $\mathcal{I}$, then a T-model for $\mathcal{C}'(\mathcal{I})$ is just a $Sign(T)$-model in $\mathcal{I}$ that satisfies T [thus $\mathbf{Mod}_{\mathcal{C}'(\mathcal{I})}(T) = \mathbf{Mod}_{\mathcal{I}}(T)$], and a T-sentence for $\mathcal{C}'(\mathcal{I})$ is a pair $\langle F\colon T'' \to T', \theta\colon T' \to T\rangle$ of morphisms of theories in $\mathcal{I}$. Finally, if A is a T-model and $\langle F, \theta\rangle$ is a T-sentence, then $A \models \langle F, \theta\rangle$ iff θA is F-free. This use of pairs of theory morphisms for constraints is perhaps more elegant than Definition 19; but theories in $\mathcal{C}'(\mathcal{I})$ do not contain the consequences in $\mathcal{I}$ of these constraints – they contain only other (given and derived) constraints. As noted above, some important applications require the new $\mathcal{I}$-sentences that follow from the added constraints.

3.3 Other Kinds of Constraint

Several variations of the data constraint notion have been proposed for the equational institution, and it seems worthwhile to generalise these variations to the level of institutions. In fact, very little of Section 3.2 depends on "F-freeness" in the definition of constraint satisfaction. This suggests weakening that notion. Recall that a Σ-model A satisfies a data constraint $c = \langle F\colon T \to T', \theta\colon Sign(T') \to \Sigma\rangle$ iff θA satisfies T' and is F-free, which means that $F^*(\theta A)$ has a free extension along F, and $(F_{F^*(\theta A)})^{\#} = \epsilon_{\theta A}\colon F^{\$}(F^*(\theta A)) \to \theta A$ is an isomorphism. For the equational institution, the most obvious ways of weakening the F-free concept are to require that $\epsilon_{\theta A}$ is only injective or only surjective, rather than bijective as for F-free. For $\epsilon_{\theta A}$ to be injective generalises the "no confusion" condition of [19]; it means that no elements of θA are identified by the natural mapping from $(F^*(\theta A))^{\$}$. Some work of Poigné uses this notion under a name like "protecting." For $\epsilon_{\theta A}$ to be surjective corresponds to what is called a "generating constraint" in [28]; it means that all elements of θA are generated by elements of $F^*(\theta A)$; this condition generalises the "no junk" condition of [19]; it is also related to various concepts of "hierarchy constraint" found in the literature, e.g., [106] and [90]. Of course, injectivity and surjectivity of $\epsilon_{\theta A}$ together imply F-freeness. Now the general notion:

Definition 25: Let $\mathcal{I}$ be an institution, let $\mathcal{A}$ denote a class of model morphisms for each signature Σ of $\mathcal{I}$, let $c = \langle F\colon T \to T', Sign(T') \to \Sigma\rangle$ be

a constraint from $\mathcal{I}$, and let A be a Σ-model from $\mathcal{I}$. Then A $\mathcal{A}$-satisfies c iff θA satisfies T' and has a free extension along F such that $(1_{F^*(\theta A)})^{\#} = \epsilon_{\theta A}$ lies in $\mathcal{A}$. (This could be further generalised to allow a different $\mathcal{A}$ for each Σ-theory T.) □

Proposition 26: Modifying the construction of $\mathcal{C}(\mathcal{I})$ to use $\mathcal{A}$-satisfaction gives an institution, and theories in this institution are as cocomplete as $\mathcal{I}$ is. □

The proofs are similar to those Proposition 23 and Theorem 24. In particular, this result implies that we can glue together theories with generating constraints using the usual colimit constructions, and we can do the specification language constructions of Clear [17]. To study generating constraints a little more closely, we assume that the morphisms in the categories of models of an institution have factorizations, in the following sense:

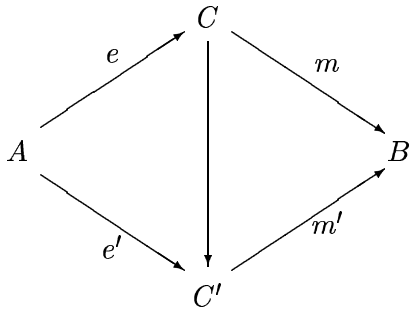
Definition 27: An **image factorization situation** for a category $\mathbf{C}$ consists of

(1) a class $\mathcal{M}$ of monics and a class $\mathcal{E}$ of epics in C such that

(2) both $\mathcal{E}$ and $\mathcal{M}$ are closed under composition,

(3) all isomorphisms are in both $\mathcal{M}$ and $\mathcal{E}$, and

(4) every morphism f can be factored as $e;m$ with $e \in \mathcal{E}$ and $m \in \mathcal{M}$ "uniquely up to isomorphism" in the sense that if $e';m'$ is another factorization of f with $e' \in \mathcal{E}$ and $m' \in \mathcal{M}$, then there is a unique isomorphism from the **center object** C of $e;m$ (i.e., the target of e and the source of m) to the center object C' of $e';m'$ such that the following diagram commutes:

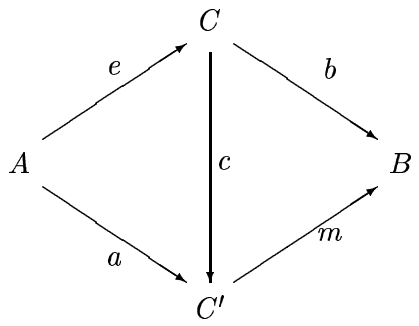


□

This concept, which seems originally due to Isbell, is actually a bit stronger than our application demands, but it has been well-studied, and has many pleasant properties; for example, it was used in [45] for a general study of some institution-like concepts. Herrlich and Strecker [62] give more detail on image factorization situations, including proofs of the following:

Proposition 28: Let $\mathcal{E}, \mathcal{M}$ be an image factorization for a category $\mathbf{C}$. Then

(1) Diagonal Fill-in Property. If $e; b = a; m$ with $e \in \mathcal{E}$ and $m \in \mathcal{M}$ and if C and C' are the center objects of the two factorizations, then there is a unique morphism $c\colon\ C \to C'$ such that $e; c = a$ and $c; m = b$.



(2) If $f \in \mathcal{E}$ and $f \in \mathcal{M}$ then f is an isomorphism.

(3) If $a; b \in \mathcal{E}$ then $b \in \mathcal{E}$, and if $a; b \in \mathcal{M}$ then $a \in \mathcal{M}$.

□

Definition 29: An institution $\mathcal{I}$ has **image factorization** iff it has an image factorization situation for each category $\mathbf{Mod}(\Sigma)$ of Σ-models.

Let $\mathcal{I}$ be a liberal institution with image factorization, let $F\colon\ T \to T'$ be a theory morphism, and let A be a T'-model. Then A is F-**generated** iff there is a morphism $e\colon\ F^{\$}(F^*(A)) \to A$ in $\mathcal{E}$ with the couniversal property.

Let $\mathcal{I}$ be an institution with image factorization, and let $F\colon\ T \to T'$ be a theory morphism. Then a T'-model A is F-**prime** iff for all $m\colon\ B \to A$ in $\mathcal{M}$ with B a T'-model and $F^*(m)\colon\ F^*(B) \to F^*(A)$ an isomorphism, m is also an isomorphism.

Let $\mathcal{I}$ be an institution with image factorization. Then a Σ-theory T is $\mathcal{M}$-**universal** iff it is "preserved under submodels" in the sense that whenever $m\colon\ B \to A$ is in $\mathcal{M}$ and A is a T-model, then B is also a T-model. □

The name "universal" is motivated by the fact that in traditional first order logic, a theory is universal iff it has a presentation with all formulae in Skolem normal form with all quantifiers universal (here, $\mathcal{M}$ consists of the injections). The following is a direct consequence of the definitions:

Fact 30: Let $\mathcal{I}$ be an institution with image factorization, let $F: T \to T'$ be a liberal theory morphism, and let A satisfy T'. Then A is F-generated iff it satisfies the $\mathcal{E}$-constraint $\langle F, 1_{Sign(T')} \rangle$. $\square$

We next show that F-generation is equivalent to F-primality under certain conditions. What we call F-primality was called F-generation by Ehrig, Wagner and Thatcher in [28][13]; our F-primality differs from their F-generation in that we replace equality by isomorphism, and their concept is restricted to the equational institution. The following result greatly increases our confidence that these concepts have been correctly captured in the institutional framework:

Theorem 31: Let $\mathcal{I}$ be a liberal institution with image factorization, and let $F: T \to T'$ be a theory morphism with T' $\mathcal{M}$-universal. Then a T'-model A is F-prime iff it is F-generated.

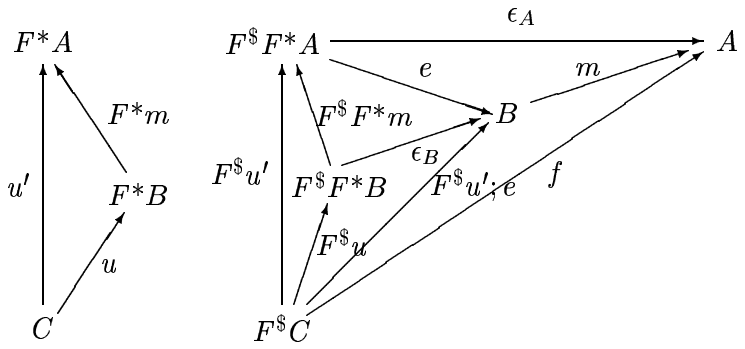
Proof: Assuming that A is F-generated, that[14] $F^*m: F^*B \to F^*A$ is an isomorphism, and that $m: B \to A$ is in $\mathcal{M}$, we must show that m is an isomorphism. We know that $\epsilon_A: F^\$(F^*A) \to A$ lies in $\mathcal{E}$. Then the desired conclusion follows from the facts that m lies in both $\mathcal{M}$ and $\mathcal{E}$, using (2) and (3) of Proposition 28, and that the left upward arrow in the diagram below is an isomorphism.

$$\begin{array}{ccc} F^\$(F^*A) & \xrightarrow{\epsilon_A} & A \\ {\scriptstyle F^\$(F^*m)}\uparrow & & \uparrow{\scriptstyle m} \\ F^\$(F^*B) & \xrightarrow[\epsilon_B]{} & B \end{array}$$

For the converse, we must show that if ϵ_A is not in $\mathcal{E}$ then there is some $m: B \to A$ in $\mathcal{M}$ not an isomorphism such that F^*m is an isomorphism. Let $e; m$ be a factorization of ϵ_A with center object B. Then m is cannot be an isomorphism, because if it were, then it would be in $\mathcal{E}$ (by (3) of Definition 27), implying that $e; m = \epsilon_A \in \mathcal{E}$ (by (2) of Definition 27), contrary to our assumption.

[13] Our terminology follows [98]; intuitively, A is F-prime if it cannot have a proper (i.e., non-isomorphic) $\mathcal{M}$-subobject $m: B \to A$ unless m is already proper when viewed through F^*.

[14] Some notation in this proof is condensed by leaving out parentheses.



To show that F^*m is an isomorphism, we will show that $\epsilon_B; m$ satisfies the universal property for ϵ_A, i.e., that given C and $f\colon F^\$C \to A$, there is a unique $u\colon C \to F^*B$ such that $F^\$u; \epsilon_B; m = f$. First, let u' be defined by the universal property of ϵ_A using the morphism f, and then let u be defined by the universal property of ϵ_B using the morphism $F^\$u'; e$. Note that B is a T'-model because T' is universal. For the uniqueness, assume $v\colon C \to F^*B$ such that $F^\$v; \epsilon_B; m = f$. Then $F^\$v; \epsilon_B = F^\$u; \epsilon_B\, (= F^\$u'; e)$, so $v = u$, by the universal property of ϵ_B. This implies that the unique morphism $h\colon F^*B \to F^*A$ such that $u' = u; h$ is an isomorphism. But F^*m is such a morphism, because ϵ is natural. □

(Note that proving F-generation implies F-prime does not require T' to be universal. We thank Andrzej Tarlecki for pointing out some errors in an earlier version of this proof; the reader may also wish to see [98] for a proof of a similar result for the case of "algebraic" institutions.)

4 Institution Morphisms — Using More than One Institution

After the work of Section 3.2, we know how to express constraints in any institution, and in particular, we can use constraints in the equational institution to specify parameterised abstract data types. We can also give loose specifications in any institution. Constraints seem most natural for liberal institutions, because we are guaranteed that satisfaction is possible. But liberality is a rather significant restriction, because non-liberal institutions can often be more expressive than liberal institutions; for example, if one adds negation to the equational institution, it ceases to be liberal; also, first order logic is not liberal. Thus, the ambitious specifier might want both the rich expressive power of first order logic and also the data structure definition power of the equational institution. This section shows he can

eat his cake and have it too. The basic idea is to precisely describe a relationship between two institutions in the form of an institution morphism, and then permit constraints that use theories from the second institution as an additional kind of sentence in this "duplex" institution; we can even have more than two institutions. Moreover, there are some other uses for institution morphisms; in particular, we will use this concept in considering when a theorem prover for one institution is sound for theories from another institution.

4.1 Institution Morphisms

Let us consider the relationship between the institution of first order logic with equality, $\mathcal{FOEQ}$, and the equational institution, $\mathcal{EQ}$. First of all, any first order signature can be reduced to an equational signature just by forgetting all its predicate symbols. Secondly, any equation can be regarded as a first order sentence just by regarding the equal sign in the equation as the distinguished binary predicate symbol that is interpreted as actual identity in models (the equal sign in the equations of the equational institution is *not* a predicate symbol, but just a punctuation symbol that separates the left and right sides). Thirdly, any first order model can be viewed as an algebra just by forgetting all its predicates. These three functors are the substance of an institution morphism $\mathcal{FOEQ} \to \mathcal{EQ}$.

Definition 32: More concrete version of Definition 13. Let $\mathcal{I}$ and $\mathcal{I}'$ be institutions. Then an **institution morphism** $\Phi : \mathcal{I} \to \mathcal{I}'$ consists of

1. a functor $\Phi : \mathbf{Sign} \to \mathbf{Sign}'$,
2. a natural transformation $\alpha : \Phi; Sen' \Rightarrow Sen$, and
3. a natural transformation $\beta : \mathbf{Mod} \Rightarrow \Phi^{op}; \mathbf{Mod}'$

such that the following **Satisfaction Condition** holds

$$m \models_\Sigma \alpha_\Sigma(e') \text{ iff } \beta_\Sigma(m) \models'_{\Phi(\Sigma)} e'$$

for any Σ-model m from $\mathcal{I}$ and any $\Phi(\Sigma)$-sentence e' from $\mathcal{I}'$. □

We leave it as an exercise for the interested reader to prove that this definition agrees with Definition 13. In fact, α is $\eta; Right$ and β is $\eta; Left$. The reader who prefers her definitions as concrete as possible may wish to see what "naturality" means in conditions 2. and 3. above: the following two diagrams spell this out, for $\phi : \Sigma \to \Sigma'$ a signature morphism in $\mathcal{I}$:

$$\begin{array}{ccc} & \overset{\alpha_\Sigma}{\longrightarrow} & \\ Sen'(\Phi(\Sigma)) & & Sen(\Sigma) \\ {\scriptstyle Sen'(\Phi(\phi))}\downarrow & & \downarrow{\scriptstyle Sen(\phi)} \\ Sen'(\Phi(\Sigma')) & \underset{\alpha_{\Sigma'}}{\longrightarrow} & Sen'(\Sigma') \end{array}$$

$$\begin{array}{ccc} & \overset{\beta_{\Sigma'}}{\longrightarrow} & \\ \mathbf{Mod}(\Sigma') & & \mathbf{Mod}'(\Phi(\Sigma')) \\ {\scriptstyle \mathbf{Mod}(\phi)}\downarrow & & \downarrow{\scriptstyle \mathbf{Mod}'(\Phi(\phi))} \\ \mathbf{Mod}(\Sigma) & \underset{\beta_\Sigma}{\longrightarrow} & \mathbf{Mod}'(\Phi(\Sigma)) \end{array}$$

The reader may now verify that $\Phi\colon \mathcal{FOEQ} \to \mathcal{EQ}$ as sketched above really is an institution morphism, by verifying the two diagrams above and the Satisfaction Condition.

Just as there is an elegant equational way to state the Satisfaction Condition for institutions (see the beginning of Section 2.3), there is also one for institution morphisms. From the basic Satisfaction Condition

$$m \models \alpha_\Sigma(e') \text{ iff } \beta_\Sigma(m) \models e'$$

for m a Σ-model from $\mathcal{I}$ and e' a $\Phi(\Sigma)$-sentence from $\mathcal{I}'$, we get

$$M \models \alpha_\Sigma(e') \text{ iff } \beta_\Sigma(M) \models e'$$

for M a collection of Σ-models, and therefore

$$\{e' \mid \beta_\Sigma(M) \models e'\} = \{e' \mid M \models \alpha_\Sigma(e')\}$$

so that

$$(\beta_\Sigma M)^* = \{e' \mid \alpha_\Sigma(e') \in M^*\}.$$

A more elegant form of this result is given in the following:

Proposition 33: Given an institution morphism $\Phi\colon \mathcal{I} \to \mathcal{I}'$, a set M of Σ-models from $\mathcal{I}$, and a set E' of $\Phi(\Sigma)$-sentences from $\mathcal{I}'$, then

1. $(\beta_\Sigma M)^* = \alpha_\Sigma^{-1}(M^*)$, and
2. $(\alpha_\Sigma E')^* = \beta_\Sigma^{-1}(E'^*)$.

Proof: The first assertion is just a reformulation of the result above this proposition. The second assertion can be proved in a similar way. □

We now turn to the question of when it is sound to use a theorem prover for one institution on (translations of) sentences from another. The basic concept is given by the following:

Definition 34: An institution morphism $\Phi : \mathcal{I} \to \mathcal{I}'$ is **sound** iff the functor β_Σ is surjective on objects, for every signature Σ from $\mathcal{I}$. □

For example, the institution morphism $\mathcal{FOEQ} \to \mathcal{EQ}$ discussed above is sound.

Proposition 35: If $\Phi : \mathcal{I} \to \mathcal{I}'$ is a sound institution morphism and if E' is a collection of Σ'-sentences from $\mathcal{I}'$, then a Σ'-sentence e' is in $E'^\bullet$ iff $\alpha_\Sigma(e')$ is in $\alpha_\Sigma(E')^\bullet$, for any signature Σ from $\mathcal{I}$ such that $\Sigma' = \Phi(\Sigma)$; that is, $E'** = \alpha_\Sigma^{-1}(\alpha_\Sigma(E')**)$ whenever $\Phi(\Sigma) = \Sigma'$.

Proof: Assuming that $\Phi(\Sigma) = \Sigma'$, the two conditions to be shown equivalent are:

(1) for all m', if $m' \models_{\Sigma'} E'$ then $m' \models_{\Sigma'} e'$.

(2) for all m, if $m \models_\Sigma \alpha_\Sigma(E')$ then $m \models_\Sigma \alpha_\Sigma(e')$.

Let us assume the second condition and prove the first from it. Thus we assume that $m' \models_{\Sigma'} E'$. Now using soundness of Φ, we get a model m from $\mathcal{I}$ such that $m' = \beta_\Sigma(m)$. Because Φ is an institution morphism, we have that $m \models_\Sigma \alpha_\Sigma(E')$. Then condition (2) gives us that $m \models_\Sigma \alpha_\Sigma(e')$. Therefore $m' \models_{\Sigma'} e'$, again because Φ is an institution morphism, and we are done.

For the converse, assume that $m \models_\Sigma \alpha_\Sigma(E')$ and let $m' = \beta_\Sigma(m)$. Then $m' \models_{\Sigma'} E'$ because Φ is an institution morphism. Now condition (1) gives us that $m' \models_{\Sigma'} e'$, and again using that Φ is an institution morphism, we conclude that $m \models_\Sigma \alpha_\Sigma(e')$. □

For example, it now follows using the institution morphism from $\mathcal{FOEQ}$ to $\mathcal{EQ}$, that a theorem prover for first order logic with equality can be used for equational theories.

Under certain rather common circumstances, an institution morphism $\Phi : \mathcal{I} \to \mathcal{I}'$ induces a functor from theories over $\mathcal{I}'$ to theories over $\mathcal{I}$.

Definition 36: Given an institution morphism $\Phi : \mathcal{I} \to \mathcal{I}'$ such that $\Phi : \mathbf{Sign} \to \mathbf{Sign}'$ has a left inverse $\Psi : \mathbf{Sign}' \to \mathbf{Sign}$ (that is, $\Psi ; \Phi = 1_{\mathbf{Sign}'}$), define $\overline{\Psi} : \mathbf{Th}_{\mathcal{I}'} \to \mathbf{Th}_{\mathcal{I}}$ by

$$\overline{\Psi}(\langle \Sigma', E' \rangle) = \langle \Psi(\Sigma'), (\alpha_{\Psi(\Sigma')}(E'))^\bullet \rangle,$$

and

$$\overline{\Psi}(\phi\colon \langle \Sigma'_0, E'_0 \rangle \to \langle \Sigma'_1, E'_1 \rangle) = \Psi(\phi)\colon \Psi(\Sigma'_0) \to \Psi(\Sigma'_1).$$

□

We now need the following:

Proposition 37: $\overline{\Psi}(\phi)$ as defined in Definition 36 is a theory morphism, and $\overline{\Psi}$ is a functor.

Proof: To show that $\overline{\Psi}(\phi)$ is a theory morphism, by the Presentation Lemma (Lemma 8), it suffices to show that

$$e \in \alpha_{\Psi(\Sigma'_0)}(E'_0) \text{ implies } \Psi(\phi)(e) \in (\alpha_{\Psi(\Sigma'_1)}(E'_1))^\bullet.$$

Hence, letting $e = \alpha_{\Psi(\Sigma'_0)}(e')$ with $e' \in E'_0$, we have to show that

$$m \models_{\Psi(\Sigma'_1)} \alpha_{\Psi(\Sigma'_1)}(E'_1) \text{ implies } m \models_{\Psi(\Sigma'_1)} \Psi(\phi)(e)$$

for all $\Psi(\Sigma'_1)$-models m. It follows that $Sen'(\phi)(e') \in E'_1$, so that

$$m \models_{\Psi(\Sigma'_1)} \alpha_{\Psi(\Sigma'_1)}(Sen'(\phi)(e')).$$

But

$$\begin{array}{rcll} \alpha_{\Psi(\Sigma'_1)}(Sen'(\phi)(e')) & = & \alpha_{\Psi(\Sigma'_1)}(Sen'(\Phi(\Psi(\phi)))(e') & \text{[because } \Psi;\Phi = 1\text{]} \\ & = & Sen(\Psi(\phi))(\alpha_{\Psi(\Sigma'_0)}(e')) & \text{[by naturality]} \\ & = & Sen(\Psi(\phi))(e), & \end{array}$$

so that $m \models_{\Psi(\Sigma'_1)} \Psi(\phi)(e)$, as desired.

We omit the proof that $\overline{\Psi}$ is functorial. □

We now consider how putting together theories over the target institution relates to their translations in the source institution. First, we need the following:

Definition 38: An institution morphism $\Phi\colon \mathcal{I} \to \mathcal{I}'$ is [**finitely**] **cordial** iff $\Phi\colon$ **Sign** $\to$ **Sign**$'$ has a left inverse $\Psi\colon$ **Sign**$'$ $\to$ **Sign** (that is, $\Psi;\Phi = 1_{\mathbf{Sign}'}$) that is [finitely] cocontinuous. □

Theorem 39: If $\Phi\colon \mathcal{I} \to \mathcal{I}'$ is a [finitely] cordial institution morphism, then $\overline{\Psi}\colon \mathbf{Th}_{\mathcal{I}'} \to \mathbf{Th}_{\mathcal{I}}$ is [finitely] cocontinuous.

Proof: Let $D'\colon G \to \mathbf{Th}_{\mathcal{I}'}$ be a diagram with $D'_n = \langle \Sigma'_n, E'_n \rangle$, and let $\gamma'\colon D' \Rightarrow T'$ be its colimit in $\mathbf{Th}_{\mathcal{I}'}$ where $T' = \langle \Sigma', E' \rangle$. Then by the

construction of Theorem 11, Σ' is the colimit of the diagram $D'; Sign'$ in $\mathbf{Sign}'$, and

$$E' = (\bigcup_{n \in |G|} \gamma'_n(E'_n))^\bullet.$$

Now define $D\colon G \to \mathbf{Th}_{\mathcal{I}}$ to be the diagram $D'; \overline{\Psi}$. Then by assumption,

$$\gamma = \langle \Psi(\gamma'_n)\colon \Psi(\Sigma'_n) \to \Psi(\Sigma') \mid n \in |G| \rangle$$

is a colimit cone in **Sign**; let $\gamma_n = \Psi(\gamma'_n)$, $\Sigma_n = \Psi(\Sigma'_n)$, and $\Sigma = \Psi(\Sigma')$. Again by the construction of Theorem 11, $\gamma\colon D \Rightarrow T$ is also a colimit cone, where $T = \langle \Sigma, E \rangle$ and

$$E = (\bigcup_{n \in |G|} \gamma_n(E_n))^\bullet$$

where $E_n = (\alpha_{\Psi(\Sigma'_n)}(E'_n))^\bullet$.

Then it remains to show that $\overline{\Psi}(T') = T$, i.e., that

$$(\alpha_\Sigma(\bigcup_n \gamma'_n E'_n))^\bullet = (\bigcup_n \gamma_n(\alpha_{\Sigma_n} E'_n)^\bullet)^\bullet.$$

Let us denote these two $\mathcal{I}$-theories by L and R, respectively. Then

$$\begin{array}{rcl} L & = & (\bigcup_n \alpha_\Sigma(\gamma' E'_n))^\bullet \\ & = & (\bigcup_n (\gamma'_n; \alpha_\Sigma) E'_n)^\bullet \\ & = & (\bigcup_n (\alpha_{\Sigma_n}; \gamma_n) E'_n)^\bullet \end{array}$$

where the last step follows by naturality of α_{Σ_n} with respect to $\gamma_n\colon \Sigma_n \to \Sigma$, as in the diagram

$$\begin{array}{ccc} Sen'(\Phi(\Sigma_n)) & \xrightarrow{\alpha_{\Sigma_n}} & Sen(\Sigma_n) \\ {\scriptstyle Sen'(\Phi(\gamma_n))}\downarrow & & \downarrow{\scriptstyle Sen(\gamma_n)} \\ Sen'(\Phi(\Sigma)) & \xrightarrow[\alpha_\Sigma]{} & Sen(\Sigma) \end{array}$$

Thus,

$$(\bigcup_n (\alpha_{\Sigma_n}; \gamma_n) E'_n)^\bullet \subseteq R,$$

so that $L \subseteq R$. Therefore, it remains to show that $R \subseteq L$. For this, we calculate

$$\begin{array}{rcl} R & = & (\bigcup_n \gamma_n(\alpha_{\Sigma_n} E'_n)^\bullet)^\bullet \\ & \subseteq & (\bigcup_n (\gamma_n(\alpha_{\Sigma_n} E'_n))^\bullet)^\bullet \\ & = & (\bigcup_n (\alpha_{\Sigma_n}; \gamma_n) E'_n)^\bullet \\ & = & L, \end{array}$$

where the second step follows by the Closure Lemma (Lemma 7) and the third follows by (11) of Proposition 3. □

It is worth noting that many institution morphisms that arise in practice are cordial. Also, this proof supports a stronger conclusion: $\overline{\Psi}$ preserves on theories whatever colimits Ψ preserves on signatures. Theorem 39 implies that if a large theory in the target institution is expressed as a colimit of smaller theories, then the corresponding theory in the source institution can also be so expressed. This can help with the practicalities of using a theorem prover for one institution on sentences from another, by modularising the proofs. Some experiments using OBJ as a theorem prover have exploited this viewpoint [40, 42].

4.2 Duplex Institutions

There are many specification examples where it is convenient to use more than one logical system. For example, we might want to use equational logic for specifying abstract data types, and to use first order logic, or perhaps temporal logic, to specify some programs over these types. In these cases, the more general institution should not be used for defining types, because it is not liberal. This motivates seeking an institution whose theories can contain both sentences from an institution $\mathcal{I}$ and constraints from another institution $\mathcal{I}'$. For example, $\mathcal{I}$ might be first order logic and $\mathcal{I}'$ might be equational logic. We call this approach "duplicity".

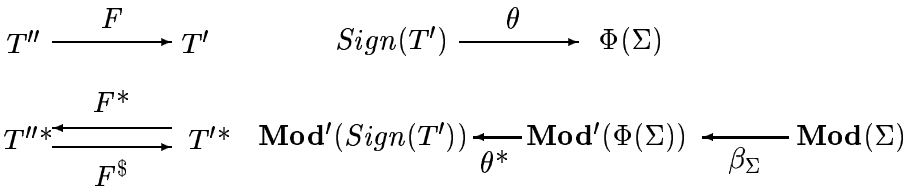
To make this construction work, we need an institution morphism $\Phi : \mathcal{I} \to \mathcal{I}'$ to express the relationship between the two institutions. This idea was introduced informally and applied to some examples in [18]. Note that all the results of this section generalise to the notion of $\mathcal{A}$-satisfaction given in Definition 25, although they are stated for the special case of data constraints, i.e., assuming $\mathcal{A}$ is the class of isomorphisms.

Definition 40: Let $\Phi : \mathcal{I} \to \mathcal{I}'$ be an institution morphism and let Σ be a signature from $\mathcal{I}$. Then a Σ-**duplex constraint** is a pair

$$c = \langle F : T'' \to T',\ \theta : Sign(T') \to \Phi(\Sigma) \rangle,$$

where F is a theory morphism from $\mathcal{I}'$ and θ is signature morphism from $\mathcal{I}'$. Then a Σ-model m from $\mathcal{I}$ **satisfies** c iff $\theta(\beta_\Sigma(m))$ satisfies T' and is F-free; as usual, we write $m \models_\Sigma c$. $\square$

The following picture of the general situation in this definition may help:



It is worth remarking that in practice, T'' and T' could be just presentations, as long as F is a theory morphism in $\mathcal{I}'$.

Definition 41: Let $\Phi : \mathcal{I} \to \mathcal{I}'$ be an institution morphism, let Σ be a signature from $\mathcal{I}$, let $c = \langle F, \theta \rangle$ be a Σ-duplex constraint, and let $\phi : \Sigma \to \Sigma'$ be a signature morphism from $\mathcal{I}$. Then the **translation** of c by ϕ is the Σ'-duplex constraint $\phi c = \langle F, \theta; \Phi(\phi) \rangle$. □

Lemma 42: Satisfaction. Let $\Phi : \mathcal{I} \to \mathcal{I}'$ be an institution morphism, let Σ be a signature from $\mathcal{I}$, let $c = \langle F, \theta \rangle$ be a Σ-duplex constraint, let $\phi : \Sigma \to \Sigma'$ be a signature morphism from $\mathcal{I}$, and let m be a Σ'-model from $\mathcal{I}$. Then

$$m \models_{\Sigma'} \phi c \text{ iff } \phi m \models_{\Sigma} c.$$

Proof: Suppose that $c = \langle F, \theta \rangle$ with $F : T'' \to T'$ and $\theta : Sign(T') \to \Phi(\Sigma)$. Then $m \models \phi c$ means that $m \models \langle F, \theta; \Phi(\phi) \rangle$, which means that $\theta(\Phi(\phi)(\beta_{\Sigma'}(m)))$ satisfies T' and is F-free. On the other hand, $\phi m \models c$ means that $\theta(\beta_{\Sigma}(\phi m))$ satisfies T and is F-free. To show these are the same, we need that $\Phi(\phi)(\beta_{\Sigma'}(m)) = \beta_{\Sigma}(\phi m)$, which is just the naturality of β (the second diagram after Definition 32). □

Definition 43: Let $\Phi : \mathcal{I} \to \mathcal{I}'$ be an institution morphism and let Σ be a signature from $\mathcal{I}$. Then the **duplex institution** over Φ, denoted $\mathcal{D}(\Phi)$ has: its signatures those from $\mathcal{I}$; its Σ-sentences the Σ-sentences from $\mathcal{I}$ plus the Σ-duplex constraints; its Σ-models the Σ-models from $\mathcal{I}$; and satisfaction is defined separately for sentences from $\mathcal{I}$ and for duplex constraints. □

For example, if we let $\Phi : \mathcal{FOEQ} \to \mathcal{EQ}$ as above, then we can write loose specifications in full first order logic, and at the same time we can impose constraints for defining data structures in the liberal institution of equational logic. This actually gives quite a powerful framework for specification, as shown by some examples written in Clear [18].

Theorem 44: If $\mathcal{I}$ is an institution and $\Phi : \mathcal{I} \to \mathcal{I}'$ is an institution morphism, then $\mathcal{D}(\Phi)$ is an institution. □

The proof is similar to that of Proposition 23 and is omitted here. By Theorem 11, this result implies

Theorem 45: If the category of signatures of $\mathcal{I}$ is [finitely] cocomplete, then so is the category $\mathbf{Th}_{\mathcal{D}(\Phi)}$ of theories of the duplex institution $\mathcal{D}(\Phi)$. □

Thus, we can do parameterised specification with constraining theories that use both sentences from $\mathcal{I}$ and duplex constraints constructed with $\mathcal{I}'$, because these are the theories over the duplex institution $\mathcal{D}(\Phi)$. Of course, we can also use the more general notion of $\mathcal{A}$-satisfaction of constraints.

There is another much simpler way to use an institution morphism $\Phi : \mathcal{I} \to \mathcal{I}'$ to construct a new institution in which one simply permits sentences from either $\mathcal{I}$ or $\mathcal{I}'$. For example, it may be convenient to use already existing equational theories when constructing new first order theories.

Definition 46: Let $\Phi : \mathcal{I} \to \mathcal{I}'$ be an institution morphism. Then $\mathcal{T}(\Phi)$ is the institution with: its signatures Σ those from $\mathcal{I}$; its Σ-sentences either Σ-sentences from $\mathcal{I}$, or else pairs of the form $p = \langle T', \theta : Sign(T') \to \Phi(\Sigma) \rangle$, where T' is a theory from $\mathcal{I}'$ and θ is a signature morphism; the Σ-models of $\mathcal{T}(\Phi)$ are those of $\mathcal{I}$; and p is **satisfied** by a Σ-model m iff $\theta(\beta_\Sigma(m))$ satisfies T. □

This construction differs from that of Definition 43 primarily in the notion of satisfaction involved: in $\mathcal{D}(\Phi)$, satisfaction of a pair $\langle F : T'' \to T', \theta : Sign(T') \to \Phi(\Sigma) \rangle$ is constraint satisfaction, involving F-freeness, whereas in $\mathcal{T}(\Phi)$, satisfaction of $\langle T', \theta : Sign(T') \to \Phi(\Sigma) \rangle$ is simple satisfaction in $\mathcal{I}'$.

Proposition 47: $\mathcal{T}(\Phi)$ is an institution. □

4.3 Multiplex Institutions

In order to use several different institutions at once, some for various kinds of constraints and others for additional expressive power, all we need is a morphism to each from some fixed base institution. Let $\mathcal{I}$ be an institution and let $\Phi_i : \mathcal{I} \to \mathcal{I}_i$ be institution morphisms for $1 \leq i \leq m + n$. Now define $\mathcal{M}(\Phi_1, \ldots, \Phi_n; \Phi_{n+1}, \ldots, \Phi_{n+m})$ to be the institution with: signatures those from $\mathcal{I}$; sentences either those from $\mathcal{I}$, or else constraints $\langle F : T'' \to T', \theta : Sign(T') \to \Phi_i(\Sigma) \rangle$ with θ, T'', T' from $\mathcal{I}_i$ for some $1 \leq i \leq n$ or else pairs $\langle T', \theta : Sign(T') \to \Phi_i(\Sigma) \rangle$ with θ, T' from $\mathcal{I}_i$ for some $n + 1 \leq i \leq n + m$; with models those of $\mathcal{I}$; and with satisfaction as usual

for the $\mathcal{I}$-sentences, for the constraints as in Definition 40, and as in Definition 46 for the others. Notice that the institutions $\mathcal{I}_i$ for $i = 1, \ldots, n$ could each use a different collection $\mathcal{A}_i$ of morphisms to define constraint satisfaction, even if they use the same institution. Thus, we could glue together first order theories that use combinations of generating constraints, hierarchy constraints, and data constraints in the equational institution, as well as loose first order axioms.

For example, consider the diagram in the category of institutions, with $\Phi_1 : \mathcal{FOEQ} \to \mathcal{EQ}$ and $\Phi_2 : \mathcal{FOEQ} \to \mathcal{HCL}$, where $\mathcal{FOEQ}$ is first order logic with equality and $\mathcal{HCL}$ is Horn clause logic. Then we can define a data type, such as **STRING-OF-NAT**, using initiality in $\mathcal{EQ}$, and inductively define a predicate, such as *even* in $\mathcal{HCL}$, e.g., by

$$even(0)$$
$$even(n) \Rightarrow even(n+2).$$

These both use data constraints, and the institutions are liberal (for $\mathcal{HCL}$, "data" is a misnomer, because we are inductively defining predicates, not data). We can then define extra functions and predicates (which are interpreted "loosely") in $\mathcal{FOEQ}$, using existential quantifiers and negation if convenient.

5 Summary and Future Research

We have formalised the intuitive notion of "logical system" or "abstract model theory" with the institution concept, and have shown that institutions whose signatures have finite colimits are useful in programming methodology. Many properties have been proved that relate classes of models and of sentences. We have also shown how to define abstract data types by several kinds of "constraint," which are abstract induction principles, including data and generating constraints. Liberal institutions, where the forgetful functors induced by theory morphisms have corresponding free functors, are particularly suitable for constraints. We have also introduced the notion of an institution morphism and shown how to use it in determining (for example) when a theorem prover for one institution can be soundly used on (translations of) theories from another institution. Institution morphisms were also used in defining duplex and multiplex institutions, which formalise the simultaneous use of more than one institution; this permits combining the expressive capabilities of several institutions. We showed that many institution morphisms preserve finite colimits of theories, which means that they preserve theory structuring mechanisms. These mechanisms support generic modules for logical programming languages such as (pure) Prolog,

OBJ and Eqlog, as well as specification languages like Clear, and program development systems like LIL.

This paper has dealt with sets of abstract sentences, using satisfaction to define the closure of a set of sentences, but has said little about proofs. In fact, it is attractive to include proofs in the notion of institution, by letting an inference step be an arrow from a tuple of sentences to another sentence. Such arrows can be composed, yielding a category, as in Lawvere theories. Then $Sen(\Sigma)$ becomes a category, and Sen takes a signature morphism covariantly to a functor ([47] gives the definition and some results). Proof theory has recently taken on new interest in logic programming and type theory (in the sense of Martin-Löf) because of doctrines like "computation is deduction" and "constructive proofs yield programs." This provides further motivation for studying institutions with proofs.

Some problems mentioned but not discussed in detail in this paper include the following:

1. It can be tedious to show that something really is an institution, particularly to check the Satisfaction Condition. This has led us to develop a method for constructing institutions from simpler structures called "charters," in a way that guarantees the Satisfaction Condition [47]. Charters are more concrete than institutions, because they specify the syntax of sentences. Parchments go even further in this direction [47].

2. An open problem mentioned in a preliminary version of this paper [44], to characterise when institutions are liberal and relate this to work of Mahr and Markowsky [74], has been solved for "strongly liberal" institutions (where $F^{\$}A$ is F-generated) by Tarlecki [98].

3. More thorough explorations of the properties of the category of institutions, including some completeness results appear in [47] and [97]; see also [100].

We conclude this section by mentioning two broad research programs that arise naturally from this paper. One is to extend the range of constructions treated institutionally; for example, Sannella and Tarlecki [88] discuss "observational equivalence," i.e., abstract machines, in an arbitrary institution. We recommend doing as much Computing Science as possible in as general an institution as possible. The second program is to carry out as much abstract model theory as possible in the more general framework of institutions; Tarlecki [97] makes a good beginning.

A Examples of Institutions

The first section of this appendix presents the details needed to establish equational logic (i.e., many-sorted general algebra) as an institution. Then, using this material, the second section develops first order logic and some related institutions. Not everything here is as elementary as it may seem; in particular, there are subtle points to the definitions of variable, equation and first order model morphism.

A.1 General Algebra

This section is a quick review of many-sorted general algebra, which provides a first example of the institution concept, and also aids in working out the details of several other institutions. If you know all about general algebra, you can skip to Section A.2. (A more leisurely exposition will appear in [42].)

We use the notation of [37] (see also [54, 42]) based on "indexed sets," in contrast to the more complex notations of [63], [8] and [10]. If I is a set (of "indices"), then an I-**indexed set** A (also called a family of sets indexed by I) is just an assignment of a set A_i to each **index** i in I. If A and B are I-indexed sets, then a **mapping**, **map**, or **morphism** of I-indexed sets, $f : A \to B$, is just an I-indexed family of functions, $f_i : A_i \to B_i$ for $i \in I$. There is an obvious composition of I-indexed mappings, $(f;g)_i = f_i;g_i$ (where the composition $f;g$ of functions f, g is defined by $(f;g)(x) = g(f(x))$). This gives a category $\mathbf{Set}_I$ of I-indexed sets. We may use the notations $A = \langle A_i \mid i \in I \rangle$ for an I-indexed set with components A_i and $f = \langle f_i : A_i \to B_i \mid i \in I \rangle$ for an I-indexed mapping $A \to B$ of I-indexed sets, where $A = \langle A_i \mid i \in I \rangle$ and $B = \langle B_i \mid i \in I \rangle$. All the basic concepts of set theory extend component-wise to I-indexed sets. Thus, $A \subseteq B$ means that $A_i \subseteq B_i$ for each i in I, $A \cap B = \langle A_i \cap B_i \mid i \in I \rangle$, etc. This can greatly simplify notation.

A.1.1 Equational Signatures

Intuitively speaking, an equational signature declares some sort symbols (to serve as names for the various kinds of data) and some operator symbols (to serve as names for functions on data), each with a declaration that gives a list of input sorts and one output sort. Then a morphism between signatures should map sorts to sorts and operators to operators, preserving their input and output sorts.

Definition 48: An **equational signature** is a pair $\langle S, \Sigma \rangle$, where S is a set (of **sort** names) and Σ is a family of sets (of **operator** names), indexed

by $S^* \times S$; we will often write just Σ instead of $\langle S, \Sigma \rangle$. σ in $\Sigma_{u,s}$ is said to have **arity** u, **sort** s, and **rank** u, s; we may write $\sigma : u \to s$ to indicate this. □

Definition 49: An **equational signature morphism** ϕ from a signature $\langle S, \Sigma \rangle$ to a signature $\langle S', \Sigma' \rangle$ is a pair $\langle f, g \rangle$ consisting of a map $f : S \to S'$ of sorts and an $S \times S^*$-indexed family of maps $g_{u,s} : \Sigma_{u,s} \to \Sigma'_{f^*(u),f(s)}$ on operator symbols, where $f^* : S^* \to S'^*$ is the extension of f to strings[15]. We will sometimes write $\phi(s)$ for $f(s)$, $\phi(u)$ for $f^*(u)$, and $\phi(\sigma)$ or even $\phi\sigma$ for $g_{u,s}(\sigma)$ when $\sigma \in \Sigma_{u,s}$. □

In the language of programming methodology, a signature declares the syntactic interface for a module, package, capsule, object, abstract machine, abstract data type, ... (unfortunately, there is much variation of terminology in this area). Signature morphisms are useful for expressing the *binding* of an actual parameter to the formal parameter of a parameterised software module.

Definition 50: The **category of equational signatures**, denoted **Sig**, has equational signatures as its objects, and has equational signature morphisms as its morphisms. The identity morphism on $\langle S, \Sigma \rangle$ is the corresponding pair of identity maps, and the composition of morphisms is the composition of their corresponding components as maps. (This clearly forms a category.) □

For the usual term algebra construction to yield an initial algebra, it is necessary that whenever $\sigma \in \Sigma_{\lambda,s}$ and $\sigma \in \Sigma_{\lambda,s'}$, then $s = s'$. We assume that all signatures satisfy this assumption. (For practical purposes, it suffices to assume that expressions have been disambiguated by a parser, so that each operator symbol has a unique sort.)

A.1.2 Algebras

Given a signature Σ, a Σ-algebra interprets each sort symbol as a set and each operator symbol as a function. Algebras, in the intuition of programming, correspond to concrete data types, i.e., to data representations in the sense of [64].

Definition 51: Let $\langle S, \Sigma \rangle$ be a signature. Then a Σ-**algebra** A is an S-indexed family of sets $|A| = \langle A_s \mid s \in S \rangle$ called the **carriers** of A, together

[15] This extension is defined by: $f^*(\lambda) = \lambda$, where λ denotes the empty string and $f^*(ws) = f^*(w)f(s)$, for w in S^* and s in S.

with an $S^* \times S$-indexed family α of maps $\alpha_{u,s} \colon \Sigma_{u,s} \to [A_u \to A_s]$ for u in S^* and s in S, where $A_{s1\ldots sn} = A_{s1} \times \ldots \times a_{sn}$ and $[A \to B]$ denotes the set of all functions from A to B. (We may sometimes write A for $|A|$ and $|A|_s$ for A_s.) For $u = s1 \ldots sn$, for σ in $\Sigma_{u,s}$ and for $\langle a1, \ldots, an \rangle$ in A_u we will write $\sigma(a1, \ldots, an)$ for $\alpha_{u,s}(\sigma)(a1, \ldots, an)$ if there is no ambiguity. □

Note that if $u = \lambda$, the empty string, then $\sigma \in \Sigma_{u,s}$ denotes a constant, because $\alpha_{u,s}(\sigma)(\star)$ is an element of A_s, where $\star$ is the unique element of A_u.

Definition 52: Given a signature Σ, a Σ-**homomorphism** from a Σ-algebra $\langle A, \alpha \rangle$ to another $\langle A', \alpha' \rangle$ is an S-indexed map $f \colon A \to A'$ such that for all σ in $\Sigma_{u,s}$ and all $a = \langle a1, \ldots, an \rangle$ in A_u the **homomorphism condition**

$$f_s(\alpha(\sigma)(a1, \ldots, an)) = \alpha'(\sigma)(f_{s1}(a1), \ldots, f_{sn}(an))$$

holds, i.e., such that the following diagram commutes:

$$\begin{array}{ccc} A_u & \xrightarrow{f_u} & A'_u \\ {\scriptstyle \alpha(\sigma)}\downarrow & & \downarrow{\scriptstyle \alpha'(\sigma)} \\ A_s & \xrightarrow[f_s]{} & A'_s \end{array}$$

□

Definition 53: The **category** $\mathbf{Alg}_\Sigma$ of Σ-**algebras** has Σ-algebras as objects and Σ-homomorphism as morphisms; composition and identity in $\mathbf{Alg}_\Sigma$ are composition and identity as maps. (This clearly forms a category). □

Alg extends to a functor on the category **Sig** of signatures, associating with each signature Σ the category $\mathbf{Alg}(\Sigma)$ of all Σ-algebras, and also defining the effect of signature morphisms on algebras:

Definition 54: The functor $\mathbf{Alg} \colon \mathbf{Sig} \to \mathbf{Cat}^{op}$ sends each signature Σ to the category $\mathbf{Alg}_\Sigma$ of all Σ-algebras, and sends each signature morphism $\phi = \langle f \colon S \to S', g \colon \Sigma \to \Sigma' \rangle$ to the functor $\mathbf{Alg}(\phi) \colon \mathbf{Alg}_{\Sigma'} \to \mathbf{Alg}_\Sigma$ that

1. sends a Σ-algebra $\langle A', \alpha' \rangle$ to the Σ-algebra $\langle A, \alpha \rangle$ with $A_s = A'_{f(s)}$ and $\alpha = g; \alpha'$, and
2. sends a Σ'-homomorphism $h' \colon A' \to B'$ to the Σ-homomorphism

$$\mathbf{Alg}(\phi)(h') = h \colon \mathbf{Alg}(\phi)(A') \to \mathbf{Alg}(\phi)(B') \text{ defined by } h_s = h'_{f(s)}.$$

It is often convenient to write $\phi(A')$ or $\phi A'$ for $\mathbf{Alg}(\phi)(A')$ and to write $\phi(h')$ for $\mathbf{Alg}(\phi)(h')$. $\square$

If S is the sort set of Σ, then there is a **forgetful functor** $U\colon \mathbf{Alg}_\Sigma \to \mathbf{Set}_S$ which sends each algebra to its S-indexed family of carriers, and sends each Σ-homomorphism to its underlying S-indexed map. (Functoriality follows from the fact that $U = \mathbf{Alg}(\phi)$, where ϕ is the signature inclusion $\langle S, \emptyset\rangle \to \Sigma$.)

For each S-indexed set X, there is a **free algebra** (also called a "term" or "word" algebra), denoted $T_\Sigma(X)$, with $|T_\Sigma(X)|_s$ consisting of all the Σ-terms of sort s using "variable" symbols from X; i.e., $(T_\Sigma(X))_s$ contains all the s-sorted terms with variables from X that can be constructed using operator symbols from Σ; moreover, the S-indexed set $T_\Sigma(X)$ forms a Σ-algebra in a natural way. In order to make this more precise, we first a consider special case, defining $(T_\Sigma)_s$ to be the least set of strings of symbols such that

1. $\Sigma_{\lambda,s} \subseteq T_{\Sigma,s}$ and
2. $\sigma \in \Sigma_{s1\ldots sn,s}$ and $ti \in T_{\Sigma,si}$ imply that the string $\sigma(t1,\ldots,tn)$ is in $T_{\Sigma,s}$.

Then the Σ-structure of T_Σ is given by α defined by:

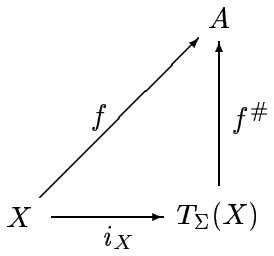
1. for $\sigma \in \Sigma_{\lambda,s}$ let $\alpha(\sigma)$ be the string σ of length 1 in $T_{\Sigma,s}$; and
2. for $\sigma \in \Sigma_{s1\ldots sn,s}$ and $ti \in T_{\Sigma,si}$ let $\alpha(\sigma)(t1,\ldots,tn)$ be the string $\sigma(t1,\ldots,tn)$ in $T_{\Sigma,s}$.

Next, define $\Sigma(X)$ to be the S-sorted signature with $(\Sigma(X))_{\lambda,s} = \Sigma_{\lambda,s} \cup X_s$ and $(\Sigma(X))_{u,s} = \Sigma_{u,s}$ if $u \neq \lambda$. Then $T_\Sigma(X)$ is just $T_{\Sigma(X)}$ regarded as a Σ-algebra rather than as a $\Sigma(X)$-algebra[16]. The freeness of $T_\Sigma(X)$ is expressed by the following:

Theorem 55: Let $i_X\colon X \to U(T_\Sigma(X))$ denote the inclusion. Then the following "universal" property holds: for any Σ-algebra A, every (S-indexed) map $f\colon X \to A$, called an **assignment**, extends uniquely to a Σ-homomorphism $f^\#\colon T_\Sigma(X) \to A$ such that $i_X; U(f^\#) = f$. $\square$

A proof may be found, for example, in [19]. We may omit the U's in such equations, as in the following traditional diagram of S-indexed sets and mappings for the above equation.

[16] I.e., $\phi(T_{\Sigma(X)})$ where $\phi\colon \Sigma \to \Sigma(X)$ is the signature inclusion.



In particular, taking $X = \emptyset$, we see that there is a unique Σ-homomorphism from T_Σ to any other Σ-algebra; thus, T_Σ is the **initial** Σ-algebra.

A.1.3 Equations and Satisfaction

We now define equations, and the satisfaction of an equation by an algebra. Let us fix, for the rest this subsection, an infinite set $\mathcal{X}$ of "variable symbols". Then a **sort assignment** is a partial function $X : \mathcal{X} \to S$ where S is a set of sorts; we also let X denote the S-indexed set defined by[17] $X_s = \{x \in \mathcal{X} \mid X(x) = s\}$. This is used in the following:

Definition 56: A Σ-**equation** e is a triple $\langle X, t1, t2\rangle$, where X is a sort assignment $\mathcal{X} \to S$ with S the set of sorts of Σ, where $t1$ and $t2$ in $|T_\Sigma(X)|_s$ are terms over (the S-sorted set) X having the same sort $s \in S$. We may write such an equation in the form $(\forall X)\ t1 = t2$. $\square$

The necessity for variable declarations in equations has been shown by Goguen and Meseguer in [50]: without them, the expected rules of deduction for many-sorted equational logic are unsound[18].

Definition 57: A Σ-algebra A **satisfies** a Σ-equation $(\forall X)\ t1 = t2$ iff $a^\#(t1) = a^\#(t2)$ for every assignment $a : X \to |A|$. In this case we write $A \models_\Sigma e$. $\square$

We now define another functor, Eqn, on the category of signatures. In order to do so, we first define for each signature morphism $\phi : \Sigma \to \Sigma'$ a function $\phi^\sim$ from Σ-terms to Σ'-terms.

Definition 58: Let $\phi : \Sigma \to \Sigma'$ be a signature morphism $\langle f : S \to S', g\rangle$, let $X : \mathcal{X} \to S$ be a sort assignment, and let X$'$ be the sort assignment $X;f$.

[17] We thank Andrzej Tarlecki for pointing out a problem with our approach to variables in [44].

[18] Some special assumptions about the form of signatures, such as that each sort has at least one constant, will also ensure soundness [50].

The following will define an S-indexed map $\phi^\sim\colon |T_\Sigma(X)| \to |\phi(T_{\Sigma'}(X'))|$: First, note that[19] $X \subseteq |\phi(T_{\Sigma'}(X'))|$ because if $x \in X_s$ then $x \in X'_{f(s)}$ and $X'_{f(s)} \subseteq |T_{\Sigma'}(X')|_{f(s)} = |\phi(T_{\Sigma'}(X'))|_s$; let $j\colon X \to |\phi(T_{\Sigma'}(X'))|$ denote this inclusion. Then j has a unique extension as a Σ-homomorphism $j^\#\colon T_\Sigma(X) \to \phi(T_{\Sigma'}(X')))$ by Theorem 55, and we simply define $\phi^\sim = |j^\#|$. □

Definition 59: The functor $Eqn\colon$ **Sig** $\to$ **Set** takes each signature Σ to the set $Eqn(\Sigma)$ of all Σ-equations, and takes each $\phi = \langle f, g\rangle\colon \Sigma \to \Sigma'$ to the function $Eqn(\phi)\colon Eqn(\Sigma) \to Eqn(\Sigma')$ defined by

$$Eqn(\phi)(\langle X, t1, t2\rangle) = \langle X; f,\ \phi^\sim(t1),\ \phi^\sim(t2)\rangle.$$

It is often convenient to write $\phi(e)$ or ϕe instead of $Eqn(\phi)(e)$. □

Proposition 60: Satisfaction Condition. If $\phi\colon \Sigma \to \Sigma'$, if e is an Σ-equation, and if A' is a Σ'-algebra, then

$$A' \models_{\Sigma'} \phi(e) \text{ iff } \phi(A') \models_\Sigma e.$$

□

We omit the proof of this result. An elegant proof using the machinery of charters and parchments may be found in [47], and a direct, but rather lengthy, proof by calculation may be found in [44]. Summarising, we have

Example 61: Many-Sorted Equational Logic is an institution, with **Sign** the category **Sig** of equational signatures (see Definition 50), with **Mod** the functor **Alg** of Definition 54, with Sen the functor Eqn of Definition 59, and with equational satisfaction as in Definition 57. The Satisfaction Condition holds by Proposition 60. We denote this institution $\mathcal{EQ}$. □

A.2 First Order Logic and Related Institutions

We follow a path like that in Section A.1, but now for some more complicated logical systems. The previous work on equational logic will greatly aid with this journey.

Definition 62: A **first order signature** Ω is a triple $\langle S, \Sigma, \Pi\rangle$, where

1. S is a set (of **sorts**),
2. Σ is an $S^* \times S$-indexed family of sets (of **operator** or **function** symbols), and

[19] Of course this means that $X_s \subseteq |\phi(T_{\Sigma'}(X'))|_s$ for each $s \in S$.

3. Π is an S^*-indexed family of sets (of **predicate** or **relation** symbols).

A **morphism of first order signatures**, from Ω to Ω', is a triple $\langle \phi_1, \phi_2, \phi_3 \rangle$, where

1. $\phi_1 : S \to S'$ is a function,
2. $\phi_2 : \Sigma \to \Sigma'$ is an $S^* \times S$-indexed family of functions $(\phi_2)_{u,s} : \Sigma_{u,s} \to \Sigma'_{\phi_1^*(u),\phi_1(s)}$ and
3. $\phi_3 : \Pi \to \Pi'$ is an S^*-indexed family of functions $(\phi_3)_u : \Pi_u \to \Pi'_{\phi_1^*(u)}$.

Let **FoSig** denote the category with first order signatures as its objects, with first order signature morphisms as its morphisms, and with the obvious identities and composition. □

Definition 63: For Ω a first order signature, an Ω-**model** (or Ω-**structure**) A consists of

1. an S-indexed family $|A|$ of non-empty sets $\langle A_s \mid s \in S \rangle$, where A_s is called the **carrier** of sort s, and
2. an $S^* \times S$-indexed family α of functions $\alpha_{u,s} : \Sigma_{u,s} \to [A_u \to A_s]$ assigning a function to each function symbol, an S^*-indexed family β of functions $\beta_u : \Pi_u \to Pow(A_u)$ assigning a relation to each predicate symbol, where $Pow(A)$ denotes the set of all subsets of A.

For $\pi \in \Pi_u$ with $u = s1 \ldots sn$ and $ai \in A_{si}$ for $i = 1, \ldots, n$, we say that "$\pi(a1, \ldots, an)$ holds" (in A) iff $(a1, \ldots, an) \in \beta(\pi)$; as usual, this may be abbreviated "$\pi(a1, \ldots, an)$."

Next, we define a **first order** Ω-**homomorphism** $f : A \to A'$ of Ω-models to be an S-indexed family of functions $f_s : A_s \to A'_s$ such that the homomorphism condition holds for Σ (as in Definition 5) and such that for $\pi \in \Pi_u$ with $u = s1 \ldots sn$, and with ai in A_{si} for $i = 1, \ldots, n$,

$$\beta(\pi)(a1, \ldots, an) \text{ implies } \beta'(\pi)(f_{si}(a1), \ldots, f_{sn}(an)).$$

(Some readers might think that "implies" above should be "iff"; however, such a notion of homomorphism would be too strong, eliminating many maps that are necessary for the term model to be initial in the Horn clause institution; see [51].)

Let **FoMod** denote the category with first order models as objects and with first order morphisms under the obvious composition. **FoMod** extends

to a functor $\mathbf{FoSig} \to \mathbf{Cat}^{op}$ as follows: Given a first order signature morphism $\phi\colon \Omega \to \Omega'$, define the functor

$$\mathbf{FoMod}(\phi)\colon \mathbf{FoMod}(\Omega') \to \mathbf{FoMod}(\Omega)$$

to send: first of all, A' in $|\mathbf{FoMod}(\Omega')|$ to $A = \phi A'$ defined by

1. $A_s = A'_{s'}$ for $s \in S$ with $s' = \phi_1(s)$,
2. $\alpha_{u,s}(\sigma) = \alpha'_{u',s'}((\phi_2)_{u,s}(\sigma))$ for $u \in S^*, s \in S$ and $\sigma \in \Sigma_{u,s}$ where $u' = \phi_1^*(u)$ and $s' = \phi_1(s)$,
3. $\beta_u(\pi) = \beta'_{u'}((\phi_3)_u(\pi))$ for $u \in S^*$ and $\pi \in \Pi_u$ with u' as above;

and secondly, to send $f'\colon A' \to B'$ in $\mathbf{FoMod}(\Omega')$ to $f = \phi f'\colon A \to B$ in $\mathbf{FoMod}(\Omega)$, where $A = \phi A'$ and $B = \phi B'$, defined by $f_s = f'_{s'}$ where $s' = \phi_1(s)$. This construction extends that of Definition 7, and it is easy to see that it does indeed give a functor. □

The next step is to define the sentences over a first order signature Ω. We do this in the usual way, by first defining terms and formulae. However, we have to be somewhat careful about variables. To this end, let $\mathcal{X}$ be a fixed infinite set of variable symbols, and let $X\colon \mathcal{X} \to S$ be a partial function, i.e., a sort assignment; as before, X also denotes the S-indexed set with $X_s = \{x \in \mathcal{X} \mid X(x) = s\}$. Now define the S-indexed family $TERM_X(\Omega)$ of (Ω, X)-**terms** to be the carriers of $T_\Sigma(X)$, the free Σ-algebra with generators X, and define $TERM(\Omega)$ to be the (disjoint) union of all the $TERM_X(\Omega)$; thus, we assume (as in the equational case) that each Ω-term comes with an explicit indication of what its variables are. Now define the (S-indexed) function *Free* on $TERM_X(\Omega)$ inductively by

1. $Free_s(x) = \{x\}$ for $x \in X_s$
2. $Free_s(\sigma(t1, \ldots, tn)) = \bigcup_{i=1}^n Free_{si}(ti)$.

Finally, *Free* extends to all of $TERM(\Omega)$.

Definition 64: A (well-formed) (Ω, X)-**formula** is an element of the carrier of the (one-sorted) free algebra $WFF_X(\Omega)$ having the **atomic** (Ω, X)-**formulae**

$$\{\pi(t1, \ldots, tn) \mid \pi \in \Pi_u \text{ with } u = s1 \ldots sn \text{ and } ti \in TERM_X(\Omega)_{si}\}$$

as generators, and having the following as its (one-sorted) signature:

1. a constant *true*,

2. a unary prefix operator $\neg$,

3. a binary infix operator &, and

4. a unary prefix operator $(\forall x)$ for each x in X.

Let $WFF(\Omega)$ be the union of all $WFF_X(\Omega)$.

The functions *Var* and *Free*, giving the sets of **variables** and of **free variables** of Ω-formulae, can be defined inductively over the above logical connectives by

1. $Var(true) = Free(true) = \emptyset$,

2. $Var(\pi(t1, \ldots, tn)) = Free(\pi(t1, \ldots, tn)) = \bigcup_{i=1}^{n} Free(ti)$,

3. $Var(\neg P) = Var(P)$, and $Free(\neg P) = Free(P)$,

4. $Var(P\&Q) = Var(P) \cup Var(Q)$, and $Free(P\&Q) = Free(P) \cup Free(Q)$, and

5. $Var((\forall x)P) = Var(P) \cup \{x\}$, and $Free((\forall x)P) = Free(P) - \{x\}$.

We now define an Ω-**sentence** to be a **closed** Ω-formula, that is, an Ω-formula P with $Free(P) = \emptyset$, and we let $FoSen(\Omega)$ denote the set of all Ω-sentences; it is the union of the sets $FoSen_X(\Omega)$ of closed (Ω, X)-formulae. For convenience, we can now define the remaining logical connectives, *false*, $\vee$, $\Rightarrow$, $\Leftrightarrow$, and $(\exists x)$ in terms of the basic ones given above in the usual way. $\square$

These definitions of $TERM_X(\Omega)$ and $WFF_X(\Omega)$ follow the "initial algebra semantics" advocated by [37] and [55] in emphasising the freeness of the algebras involved. We exploit this to define the effect of *FoSen* on first order signature morphisms, so that it becomes a functor **FoSig** $\rightarrow$ **Set**. Given $\phi\colon \Omega \rightarrow \Omega'$, we can define $FoSen_X(\phi)\colon FoSen_X(\Omega) \rightarrow FoSen_{X'}(\Omega')$, where $X' = X; \phi_1$, using initiality. Because $(\phi_1, \phi_2)\colon \Sigma \rightarrow \Sigma'$ is a signature morphism, there is an induced morphism $\psi\colon T_\Sigma(X) \rightarrow T_{\Sigma'}(X')$, which then gives us $\psi\colon TERM_X(\Omega) \rightarrow TERM_{X'}(\Omega')$. Now define $WFF_X(\phi)\colon WFF_X(\Omega) \rightarrow WFF_{X'}(\Omega')$ by its effect on the generators of $WFF_X(\Omega)$, which are the atomic formulae, by

$$WFF_X(\phi)(\pi(t1, \ldots, tn)) = \phi_3(\pi)(\psi(t1), \ldots, \psi(tn)).$$

Finally, define $FoSen_X(\phi)$ to be the restriction of $WFF_X(\phi)$ to $FoSen_X(\Omega) \subseteq WFF_X(\Omega)$. For this to work, it must be checked that $WFF_X(\phi)$ carries closed Ω-formulae to closed Ω'-formulae; but this is easy. $FoSen(\phi)$ is then

defined to be the union of the functions $FoSen_X(\phi)$ on the union of the sets $FoSen_X(\Omega)$.

It remains to define satisfaction. This corresponds to the usual "semantic definition of truth" (originally due to Tarski [102]). If A is a first order model, let $Asgn_X(A)$ denote the set of all **assignments** of values in A to variables in X, i.e, $[X \to A]$, the set of all S-indexed functions $f : X \to A$.

Definition 65: Given an Ω-sentence P and an Ω-model A, define $Asgn_X(A, P)$, the set of assignments in A for which P is true, inductively as follows:

1. if $P = \pi(t1, \ldots, tn)$ then $f \in Asgn_X(A, P)$ iff $(f^\#(t1), ..., f^\#(tn)) \in \beta(\pi)$, where $f^\#(t)$ denotes the evaluation of the Σ-term t in the Σ-algebra part of A with the values of variables given by the assignment f using initiality.
2. $Asgn_X(A, true) = Asgn_X(A)$.
3. $Asgn_X(A, \neg P) = Asgn_X(A) - Asgn_X(A, P)$.
4. $Asgn_X(A, P\&Q) = Asgn_X(A, P) \cap Asgn_X(A, Q)$.
5. $Asgn_X(A, (\forall x)P) = \{f \mid Asgn_X(A, f, x) \subseteq Asgn_X(A, P)\}$, where $Asgn_X(A, f, x)$ is the set of all assignments f' that agree with f except possibly on the variable x from X.

Then A **satisfies** $P \in WFF_\Omega(X)$, written $A \models_\Omega P$, iff $Asgn_X(A, P) = Asgn_X(A)$. □

Finally, we must verify the satisfaction condition. This follows from an argument much like that used for the equational case, and is omitted here; the result can also be obtained using the method of "charters" [47]. Thus, summarising the above, we have

Example 66: (Many-Sorted) First Order Logic is an institution; let us denote it $\mathcal{FOL}$. □

Example 67: (Many-Sorted) First Order Logic with Equality. This institution is closely related to that of Example 66. A signature for first order logic with equality is a first order signature $\Omega = \langle S, \Sigma, \Pi \rangle$ that has a particular predicate symbol $\equiv_s$ in Π_{ss} for each $s \in S$. A morphism of signatures for first order logic with equality is a morphism of signatures for first order logic that preserves these predicate symbols, i.e., that satisfies $\phi_3(\equiv_s) = \equiv_{\phi_1(s)}$ for all $s \in S$. This gives a category **FoSigEq** of signatures for first order logic with equality.

If Ω is a signature for first order logic with equality, then a model for it is just an Ω-model A in the usual first order sense satisfying the additional condition that

$$a \equiv_s a' \text{ iff } a = a'$$

for all $s \in S$ and for all $a, a' \in A_s$.

A homomorphism of first order Ω-models with equality is just a first order Ω-homomorphism (in the sense of Definition 63), and we get a category $\mathbf{FoModEq}(\Omega)$ of Ω-models for each signature Ω in $|\mathbf{FoSigEq}|$, and $\mathbf{FoModEq}$ is a functor on $\mathbf{FoSigEq}$. Ω-sentences are defined just as in Example 66, and so is satisfaction. We thus get a functor $\mathit{FoSenEq} : \mathbf{FoSigEq} \to \mathbf{Set}$. The Satisfaction Condition follows immediately from that of first order logic without equality. Let us denote this institution by $\mathcal{FOEQ}$. □

It is interesting to note that many liberal institutions that are familiar from logic have served as bases for programming languages, including those mentioned below.

Example 68: (Many-Sorted) Horn Clause Logic with Equality. We may specialise the previous example by limiting the form that sentences can take, but without restricting either the predicate or operator symbols that can enter into them. In particular, we maintain the equality symbol with its fixed interpretation from Example 67; but we require that all sentences be of the form

$$(\forall X)\ A_1 \,\&\, A_2 \,\&\, \ldots A_n \Rightarrow A,$$

where A and each A_i is an atomic formula $\pi(t1, \ldots, t_n)$ with variables from X. In particular, disjunction, negation and existential quantifiers are excluded. That this is an institution follows from the facts that first order logic with equality is an institution, and that the class of Horn clauses is closed under translation by signature morphisms. Let us denote this institution $\mathcal{HCLEQ}$; it is the basis[20] for the logical programming language Eqlog [51]. □

Example 69: (Many-Sorted) Conditional Equational Logic. As a specialisation of Example 68, let equality be the only predicate symbol. This gives the institution often called conditional equational logic. It is the basis[20] of the logical programming language OBJ [32, 56, 48, 33]. □

[20] More precisely, the order-sorted [53] variant of this logic is the basis.

Example 70: (Many-Sorted) Horn Clause Logic without Equality. We can also restrict Example 68 by dropping equality with its fixed interpretation. This too is obviously an institution because it is just a restriction of ordinary first order logic; it is the basis for the logical programming language (pure) Prolog [71]. Let us denote this institution $\mathcal{HCL}$. □

Gabriel and Ulmer [34] describe a very general notion of logical system with some results for demonstrating liberality.

It seems clear that many-sorted temporal or modal logic can be treated in much the same way, by adding the appropriate modal operators to the signature and defining their correct interpretation in all models, which will be "Kripke" or "possible world" structures. Higher order equational logic is also presumably an institution; the development should again follow that of equational logic, using higher order sorts and operator symbols[21]. "In-equational logic" [11], order-sorted equational logic [53], and various kinds of infinitary equational logic, such as continuous algebras [55, 107], are also institutions [99]. We further conjecture that in general, mechanical theorem provers, such as the Boyer-Moore prover [12], Aubin's system [3] and STP [95], are based on logical systems that are institutions (or if they are not, they should be modified so that they are!). Clearly, it would be helpful to have some general results to help establish that various logical systems are institutions. This motivates the study of "charters" and "parchments" in [47].

A way to generate many other examples [74] is to let the sentences over a signature Ω be all the specifications using Ω written in some specification language (such as Clear); we could think of such a specification as a convenient abbreviation for a (possibly very large) conjunction of simpler sentences. However, this seems inappropriate under the view that the purpose of a specification language is to assist with programming-in-the-large, by providing facilities for building new specifications by reusing old specifications as a whole, e.g., by applying a parameterised specification, as opposed to local operations on sentences, such as conjunction.

References

[1] Jean-Raymond Abrial, S.A. Schuman, and Bertrand Meyer. Specification language. In R.M. McKeag and A.M. MacNaughten, editors, *On the Construction of Programs: An Advanced Course*, pages 343–410. Cambridge, 1980.

[21] The sorts involved will be the objects of the free Cartesian closed category on the basic sort set [82, 92].

[2] Michael Arbib and Ernest Manes. *Arrows, Structures and Functors.* Academic, 1975.

[3] R. Aubin. *Mechanizing Structural Induction.* PhD thesis, University of Edinburgh, 1976.

[4] Arnon Avron, Furio Honsell, and Ian Mason. Using typed lambda calculus to implement formal systems on a computer. Technical Report ECS-LFCS-87-31, Laboratory for Computer Science, University of Edinburgh, 1987.

[5] John Barwise and Solomon Feferman. *Model-Theoretic Logics.* Springer, 1985.

[6] Jon Barwise. Axioms for abstract model theory. *Annals of Mathematical Logic*, 7:221–265, 1974.

[7] Christoph Beierle and Angelika Voss. Implementation specifications. In Hans-Jörg Kreowski, editor, *Recent Trends in Data Type Specification*, volume Informatik-Fachberichte 116, pages 39–52. Springer, 1985. Selected papers from the Third Workshop on Theory and Applications of Abstract Data Types.

[8] Jean Benabou. Structures algébriques dans les catégories. *Cahiers de Topologie et Géometrie Différentiel*, 10:1–126, 1968.

[9] Garrett Birkhoff. *Lattice Theory.* American Mathematical Society, 1948. Colloquium Publications, Volume XXV (revised edition, 1960).

[10] Garrett Birkhoff and J. Lipson. Heterogeneous algebras. *Journal of Combinatorial Theory*, 8:115–133, 1970.

[11] Steven Bloom. Varieties of ordered algebras. *Journal of Computer and System Sciences*, 13:200–212, 1976.

[12] Robert Boyer and J Moore. *A Computational Logic.* Academic, 1980.

[13] Rod Burstall. Proving properties of programs by structural induction. *Computer Journal*, 12(1):41–48, 1969.

[14] Rod Burstall. Inductively defined functions. In Hartmut Ehrig, Christiane Floyd, Maurice Nivat, and James Thatcher, editors, *Mathematical Foundations of Software Development, Volume 1*, pages 92–96. Springer, 1985. Lecture Notes in Computer Science, Volume 185.

[15] Rod Burstall and Joseph Goguen. Putting theories together to make specifications. In Raj Reddy, editor, *Proceedings, Fifth International Joint Conference on Artificial Intelligence*, pages 1045–1058. Department of Computer Science, Carnegie-Mellon University, 1977.

[16] Rod Burstall and Joseph Goguen. Semantics of Clear. Unpublished notes handed out at the 1978 Symposium on Algebra and Applications, Stefan Banach Center, Warsaw, Poland, 1978.

[17] Rod Burstall and Joseph Goguen. The semantics of Clear, a specification language. In Dines Bjorner, editor, *Proceedings, 1979 Copenhagen Winter School on Abstract Software Specification*, pages 292–332. Springer, 1980. Lecture Notes in Computer Science, Volume 86; based on unpublished notes handed out at the Symposium on Algebra and Applications, Stefan Banach Center, Warsaw, Poland.

[18] Rod Burstall and Joseph Goguen. An informal introduction to specifications using Clear. In Robert Boyer and J Moore, editors, *The Correctness Problem in Computer Science*, pages 185–213. Academic, 1981. Reprinted in *Software Specification Techniques*, Narain Gehani and Andrew McGettrick, editors, Addison-Wesley, 1985, pages 363-390.

[19] Rod Burstall and Joseph Goguen. Algebras, theories and freeness: An introduction for computer scientists. In Martin Wirsing and Gunther Schmidt, editors, *Theoretical Foundations of Programming Methodology*, pages 329–350. Reidel, 1982. Proceedings, 1981 Marktoberdorf NATO Summer School, NATO Advanced Study Institute Series, Volume C91.

[20] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum, 1978.

[21] Paul M. Cohn. *Universal Algebra*. Harper and Row, 1965. Revised edition 1980.

[22] Hans-Dieter Ehrich. On the theory of specification, implementation and parameterization of abstract data types. *Journal of the Association for Computing Machinery*, 29:206–227, 1982.

[23] Hartmut Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, Hartmut Ehrig, and Gregor Rozenberg, editors, *Graph Gramars and their Application to Computer Science and Biology*, pages 1–69. Springer, 1979. Lecture Notes in Computer Science, Volume 73.

[24] Hartmut Ehrig. Categorical concept of constraints for algebraic specifications. In Hartmut Ehrig, Horst Herrlich, Hans-Jörg Kerowski, and Gerhard Preuss, editors, *Categorical Methods in Computer Science*, pages 1–15. Springer, 1989. Lecture Notes in Computer Science, Volume 393.

[25] Hartmut Ehrig, Werner Fey, and Horst Hansen. ACT ONE: An algebraic specification language with two levels of semantics. Technical Report 83-03, Technical University of Berlin, Fachbereich Informatik, 1983.

[26] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer, 1985. EATCS Monographs on Theoretical Computer Science, Volume 6.

[27] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*. Springer, 1990. EATCS Monographs on Theoretical Computer Science, Volume 21.

[28] Hartmut Ehrig, Eric Wagner, and James Thatcher. Algebraic specifications with generating constraints. In J. Diaz, editor, *Proceedings, 10th Colloquium on Automata, Languages and Programming*, pages 188–202. Springer, 1983. Lecture Notes in Computer Science, Volume 154.

[29] Werner Fey. Pragmatics, concepts, syntax, semantics and correctness notions of ACT TWO: An algebraic module specification and interconnection language. Technical Report 88-26, Technical University of Berlin, Fachbereich Informatik, 1988.

[30] José Fiadeiro and Amilcar Sernadas. Structuring theories on consequence. In Donald Sannella and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification*, pages 44–72. Springer, 1988. Lecture Notes in Computer Science, Volume 332.

[31] José Fiadeiro, Amilcar Sernadas, and Christina Sernadas. Knowledgebases as structured theories. In K.V. Nori, editor, *Foundations of Software Technology and Theoretical Computer Science*, pages 469–486. Springer, 1987. Lecture Notes in Computer Science, Volume 287.

[32] Kokichi Futatsugi, Joseph Goguen, Jean-Pierre Jouannaud, and José Meseguer. Principles of OBJ2. In Brian Reid, editor, *Proceedings, Twelfth ACM Symposium on Principles of Programming Languages*, pages 52–66. Association for Computing Machinery, 1985.

[33] Kokichi Futatsugi, Joseph Goguen, José Meseguer, and Koji Okada. Parameterized programming in OBJ2. In Robert Balzer, editor, *Proceedings, Ninth International Conference on Software Engineering*, pages 51–60. IEEE Computer Society, March 1987.

[34] P. Gabriel and F. Ulmer. *Lokal Präsentierbare Kategorien*. Springer, 1971. Lecture Notes in Mathematics, Volume 221.

[35] Joseph Goguen. Mathematical representation of hierarchically organized systems. In E. Attinger, editor, *Global Systems Dynamics*, pages 112–128. S. Karger, 1971.

[36] Joseph Goguen. Categorical foundations for general systems theory. In F. Pichler and R. Trappl, editors, *Advances in Cybernetics and Systems Research*, pages 121–130. Transcripta Books, 1973.

[37] Joseph Goguen. Semantics of computation. In Ernest G. Manes, editor, *Proceedings, First International Symposium on Category Theory Applied to Computation and Control*, pages 234–249. University of Massachusetts at Amherst, 1974. Also in Lecture Notes in Computer Science, Volume 25, Springer, 1975, pages 151-163.

[38] Joseph Goguen. Parameterized programming. *Transactions on Software Engineering*, SE-10(5):528–543, September 1984.

[39] Joseph Goguen. Reusing and interconnecting software components. *Computer*, 19(2):16–28, February 1986. Reprinted in *Tutorial: Software Reusability*, Peter Freeman, editor, IEEE Computer Society, 1987, pages 251-263, and in *Domain Analysis and Software Systems Modelling*, Ruben Prieto-Diaz and Guillermo Arango, editors, IEEE Computer Society, 1991, pages 125-137.

[40] Joseph Goguen. OBJ as a theorem prover, with application to hardware verification. In V.P. Subramanyan and Graham Birtwhistle, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 218–267. Springer, 1989. Also Technical Report SRI-CSL-88-4R2, SRI International, Computer Science Lab, August 1988.

[41] Joseph Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67, March 1991. Also, Programming Research Group Technical Monograph PRG-72, University of Oxford, March 1989.

[42] Joseph Goguen. *Theorem Proving and Algebra*. Prentice-Hall, to appear 1992. Book in preparation.

[43] Joseph Goguen and Rod Burstall. Some fundamental properties of algebraic theories: a tool for semantics of computation. Technical report, Deptartment of Artificial Intelligence, University of Edinburgh, 1978. DAI Research Report Number 5; expended version appears in *Theoretical Computer Science, 31*, pp. 175-209, 1984.

[44] Joseph Goguen and Rod Burstall. Introducing institutions. In Edward Clarke and Dexter Kozen, editors, *Proceedings, Logics of Programming Workshop*, pages 221–256. Springer, 1984. Lecture Notes in Computer Science, Volume 164.

[45] Joseph Goguen and Rod Burstall. Some fundamental algebraic tools for the semantics of computation, part 2: Signed and abstract theories. *Theoretical Computer Science*, 31(3):263–295, 1984.

[46] Joseph Goguen and Rod Burstall. Some fundamental algebraic tools for the semantics of computation, part 1: Comma categories, colimits, signatures and theories. *Theoretical Computer Science*, 31(2):175–209, 1984.

[47] Joseph Goguen and Rod Burstall. A study in the foundations of programming methodology: Specifications, institutions, charters and parchments. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Proceedings, Conference on Category Theory and Computer Programming*, pages 313–333. Springer, 1986. Lecture Notes in Computer Science, Volume 240; also, Report Number CSLI-86-54, Center for the Study of Language and Information, Stanford University, June 1986.

[48] Joseph Goguen, Claude Kirchner, Hélène Kirchner, Aristide Mégrelis, and José Meseguer. An introduction to OBJ3. In Jean-Pierre Jouannaud and Stephane Kaplan, editors, *Proceedings, Conference on Conditional Term Rewriting*, pages 258–263. Springer, 1988. Lecture Notes in Computer Science, Volume 308.

[49] Joseph Goguen and José Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In M. Nielsen and E.M. Schmidt, editors, *Proceedings, 9th International Conference on Automata, Languages and Programming*, pages 265–281. Springer, 1982. Lecture Notes in Computer Science, Volume 140.

[50] Joseph Goguen and José Meseguer. Completeness of many-sorted equational logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985. Preliminary versions have appeared in: *SIGPLAN Notices*, July 1981, Volume 16, Number 7, pages 24-37; SRI Computer Science Lab Technical Report CSL-135, May 1982; and Report CSLI-84-15, Center for the Study of Language and Information, Stanford University, September 1984.

[51] Joseph Goguen and José Meseguer. Eqlog: Equality, types, and generic modules for logic programming. In Douglas DeGroot and Gary Lindstrom, editors, *Logic Programming: Functions, Relations and Equations*, pages 295–363. Prentice-Hall, 1986. An earlier version appears in *Journal of Logic Programming*, Volume 1, Number 2, pages 179-210, September 1984.

[52] Joseph Goguen and José Meseguer. Unifying functional, object-oriented and relational programming, with logical semantics. In Bruce Shriver and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. MIT, 1987. Preliminary version in *SIGPLAN Notices*, Volume 21, Number 10, pages 153-162, October 1986.

[53] Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Technical Report SRI-CSL-89-10, SRI International, Computer Science Lab, July 1989. Given as lecture at Seminar on Types, Carnegie-Mellon University, June 1983; many draft versions exist.

[54] 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 T.J. Watson Research Center, October 1976. In *Current Trends in Programming Methodology, IV*, Raymond Yeh, editor, Prentice-Hall, 1978, pages 80-149.

[55] 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, January 1977. An early version is "Initial Algebra Semantics", with James Thatcher, IBM T.J. Watson Research Center Report RC 4865, May 1974.

[56] Joseph Goguen and Timothy Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, SRI International, Computer Science Lab, August 1988. Revised version to appear with additional authors José Me-

seguer, Kokichi Futatsugi and Jean-Pierre Jouannaud, in *Applications of Algebraic Specification using OBJ*, edited by Joseph Goguen, Derek Coleman and Robin Gallimore, Cambridge, 1992.

[57] Robert Goldblatt. *Topoi, the Categorial Analysis of Logic.* North-Holland, 1979.

[58] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *Proceedings, Second Symposium on Logic in Computer Science*, pages 194–204. IEEE Computer Society, 1987.

[59] Robert Harper, David MacQueen, and Robin Milner. Standard ML. Technical Report ECS-LFCS-86-2, Department of Computer Science, University of Edinburgh, 1986.

[60] Robert Harper, Donald Sannella, and Andrzej Tarlecki. Structure and representation in LF. Technical Report ECS-LFCS-89-75, Laboratory for Computer Science, University of Edinburgh, 1987.

[61] Ian Hayes. *Specification Case Studies.* Prentice-Hall, 1987.

[62] Horst Herrlich and George Strecker. *Category Theory.* Allyn and Bacon, 1973.

[63] P.J. Higgins. Algebras with a scheme of operators. *Mathematische Nachrichten*, 27:115–132, 1963.

[64] C.A.R. Hoare. Proof of correctness of data representation. *Acta Informatica*, 1:271–281, 1972.

[65] H. Kaphengst and Horst Reichel. Algebraische algorithemtheorie. Technical Report WIB 1, VEB Robotron, Zentrum fur Forschung und Technik, Dresden, 1971. In German.

[66] Joachim Lambek and Phil Scott. *Introduction to Higher Order Categorical Logic.* Cambridge, 1986. Cambridge Studies in Advanced Mathematics, Volume 7.

[67] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer, 1971.

[68] Saunders Mac Lane and Garrett Birkhoff. *Algebra.* Macmillan, 1967.

[69] F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings, National Academy of Sciences, U.S.A.*, 50:869–872, 1963. Summary of Ph.D. Thesis, Columbia University.

[70] F. William Lawvere. An elementary theory of the category of sets. *Proceedings, National Academy of Sciences, U.S.A.*, 52:1506–1511, 1964.

[71] John Lloyd. *Foundations of Logic Programming.* Springer, 1984.

[72] Michael Lowry. Institutionalizing problem reformulation. In Paul Benjamin, Indur Mandhyan, and Ernest Manes, editors, *Proceedings, Workshop on Category Theory in AI and Robotics*, pages 159–194. Philips Laboratories, 1989.

[73] David MacQueen and Donald Sannella. Completeness of proof systems for equational specifications. *IEEE Transactions on Software Engineering*, SE-11(5):454–461, May 1985.

[74] Bernd Mahr and Johann Makowsky. An axiomatic approach to semantics of specification languages. In *Proceedings, 6th GI Conference on Theoretical Computer Science.* Springer, 1983. Lecture Notes in Computer Science, Volume 145.

[75] Bernd Mahr and Johann Makowsky. Characterizing specification languages which admit initial semantics. *Theoretical Computer Science*, 31:49–60, 1984.

[76] Brian Mayoh. Galleries and institutions. Technical Report DAIMI PB-191, Aarhus University, 1985.

[77] John McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1,2):27–39, 1980.

[78] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium, 1987.* North-Holland, 1989.

[79] José Meseguer and Joseph Goguen. Initiality, induction and computability. In Maurice Nivat and John Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge, 1985.

[80] Peter Mosses. Unified algebras and institutions. Technical Report DAIMI PB-274, Computer Science Department, Aarhus University, 1989.

[81] Fernando Orejas, V. Sacristán, and S. Clerici. Development of algebraic specifications with constraints. In Hartmut Ehrig, Horst Herrlich, Hans-Jörg Kerowski, and Gerhard Preuss, editors, *Categorical Methods in Computer Science*, pages 102–123. Springer, 1989. Lecture Notes in Computer Science, Volume 393.

[82] Kamran Parsaye-Ghomi. *Higher Order Data Types*. PhD thesis, UCLA, Computer Science Department, January 1982.

[83] Axel Poigné. Foundations are rich institutions, but institutions are poor foundations. In Hartmut Ehrig, Horst Herrlich, Hans-Jörg Kerowski, and Gerhard Preuss, editors, *Categorical Methods in Computer Science*, pages 82–101. Springer, 1989. Lecture Notes in Computer Science, Volume 393.

[84] Horst Reichel. Initially restricting algebraic theories. In Piotr Dembinski, editor, *Mathematical Foundations of Computer Science*, pages 504–514. Springer, 1980. Lecture Notes in Computer Science, Volume 88.

[85] Horst Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras*. Clarendon, 1987.

[86] Raymond Reiter. On closed world data bases. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum, 1978.

[87] Donald Sannella and Andrzej Tarlecki. Extended ML: an institution independent framework for formal program development. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Proceedings, Summer Workshop on Category Theory and Computer Programming*, pages 364–389. Springer, 1986. Lecture Notes in Computer Science, Volume 240.

[88] Donald Sannella and Andrzej Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Science*, 34:150–178, 1987. Earlier version in *Proceedings, Colloquium on Trees in Algebra and Programming*, Lecture Notes in Computer Science, Volume 185, Springer, 1985.

[89] Donald Sannella and Andrzej Tarlecki. Specifications in an arbitrary institution. *Information and Control*, 76:165–210, 1988. Earlier version in *Proceedings, International Symposium on the Semantics of Data Types*, Lecture Notes in Computer Science, Volume 173, Springer, 1985.

[90] Donald Sannella and Martin Wirsing. Implementations of parameterized specifications. In M. Nielsen and E.M. Schmidt, editors, *Proceedings, Ninth Colloquium on Automata, Languages and Programming*, pages 473–488. Springer, 1982. Lecture Notes in Computer Science, Volume 140.

[91] Dana Scott. Completeness and axiomatizability in many-valued logic. In Leon Henkin *et al.*, editor, *Proceedings, Tarski Symposium*, pages 411–435. American Mathematical Society, 1974.

[92] Dana Scott. Relating theories of the lambda calculus. In J. P. Seldin and J. R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 404–450. Academic, 1980.

[93] Amilcar Sernadas and Christina Sernadas. Conceptual modelling for knowledge-based DSS development. In C. Holsapple and A. Whinston, editors, *Decision Support Systems: Theory and Application*, pages 91–135. Springer, 1987.

[94] John C.C. Shepardson. Negation as failure. *Journal of Logic Programming*, 1(1):51–79, 1984.

[95] Rob Shostak, Richard Schwartz, and P. Michael Melliar-Smith. STP: A mechanized logic for specification and verification. Technical report, Computer Science Lab, SRI International, 1981.

[96] J. Michael Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall, 1989.

[97] Andrzej Tarlecki. Bits and pieces of the theory of institutions. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Proceedings, Summer Workshop on Category Theory and Computer Programming*, pages 334–360. Springer, 1986. Lecture Notes in Computer Science, Volume 240.

[98] Andrzej Tarlecki. On the existence of free models in abstract algebraic institutions. *Theoretical Computer Science*, 37:269–304, 1986. Preliminary version, University of Edinburgh, Computer Science Department Technical Report CSR-165-84, 1984.

[99] Andrzej Tarlecki. Quasi-varieties in abstract algebraic institutions. *Journal of Computer and System Sciences*, 33(3):333–360, 1986. Original version, University of Edinburgh Technical Report CSR-173-84.

[100] Andrzej Tarlecki, Rod Burstall, and Joseph Goguen. Some fundamental algebraic tools for the semantics of computation, part 3: Indexed categories. Technical Report PRG-77, Programming Research Group, Oxford University, August 1989. To appear in *Theoretical Computer Science.*

[101] Alfred Tarski. Fundamentale begriffe der methodologie der deduktiven wissenschaften, 1930.

[102] Alfred Tarski. The semantic conception of truth. *Philos. Phenomenological Research*, 4:13–47, 1944.

[103] James Thatcher, Eric Wagner, and Jesse Wright. Data type specification: Paramerization and the power of specification techniques. In *Proceedings, Sixth Symposium on Principles of Programming Languages*. Association for Computing Machinery, 1979. Also in *TOPLAS 4*, pages 711-732, 1982.

[104] William Joseph Tracz. *Formal Specification of Parameterized Programs in* LILLEANNA. PhD thesis, Stanford University, to appear 1992.

[105] Martin Wirsing. Structured algebraic specifications: A kernel language. *Theoretical Computer Science*, 42:123–249, 1986.

[106] Martin Wirsing, Peter Pepper, W. Partsch, N. Dosch, and Manfred Broy. On hierarchies of abstract data types. *Acta Informatics*, 20:1–33, 1983.

[107] Jesse Wright, James Thatcher, Eric Wagner, and Joseph Goguen. Rational algebraic theories and fixed-point solutions. In Michael J. Fischer, editor, *Proceedings, Seventeenth Symposium on Foundations of Computing*, pages 147–158. IEEE, 1976.

Contents