Introducing Institutions

J. A. Goguen[1] and R. M. Burstall
SRI International and the University of Edinburgh

Abstract

There is a population explosion among the logical systems being used in computer science. Examples include first order logic (with and without equality), equational logic, Horn clause logic, second order logic, higher order logic, infinitary logic, dynamic logic, process logic, temporal logic, and modal logic; moreover, there is a tendency for each theorem prover to have its own idiosyncratic logical system. Yet it is usual to give many of the same results and applications for each logical system; of course, this is natural in so far as there are basic results in computer science that are independent of the logical system in which they happen to be expressed. But we should not have to do the same things over and over again; instead, we should generalize, and do the essential things once and for all! Also, we should ask what are the relationships among all these different logical systems. This paper shows how some parts of computer science can be done in any suitable logical system, by introducing the notion of an **institution** as a precise generalization of the informal notion of a "logical system." A first main result shows that if an institution is such that interface declarations expressed in it can be glued together, then **theories** (which are just sets of sentences) in that institution can also be glued together. A second main result gives conditions under which a theorem prover for one institution can be validly used on theories from another; this uses the notion of an institution morphism. A third main result shows that institutions admiting free models can be extended to institutions whose theories may include, in addition to the original sentences, various kinds of constraints upon interpretations; such constraints are useful for defining abstract data types, and include so-called "data," "hierarchy," and "generating" constraints. Further results show how to define insitutions that mix sentences from one institution with constraints from another, and even mix sentences and (various kinds of) constraints from several different institutions. It is noted that general results about institutions apply to such "multiplex" institutions, including the result mentioned above about gluing together theories. Finally, this paper discusses some applications of these results to specification languages, showing that much of that subject is in fact independent of the institution used.

1 Introduction

Recent work in programming methodology has been based upon numerous different logical systems. Perhaps most popular are the many variants of first order logic found, for example, in the theorem provers used in various program verification projects. But also popular are equational logic, as used in the study of abstract data types, and first order Horn clause logic, as used in "logic programming." More exotic logical systems, such as temporal logic, infinitary logic, and continuous algebra have also been proposed to handle features such as concurrency and non-termination, and most systems exist in both one-sorted and many-sorted forms. However, it seems apparent that much of programming methodology is actually *completely*

[1]Research supported in part by Office of Naval Research contracts N00014-80-0296 and N00014-82-C-0333, and National Science Foundation Grant MCS8201380.

independent of what underlying logic is chosen. In particular, if [Burstall & Goguen 77] are correct that the essential purpose of a specification language is to say how to put (small and hopefully standard) theories together to make new (and possibly very large) specifications, then much of the syntax and semantics of specification does not depend upon the logical system in which the theories are expressed; the same holds for implementing a specification, and for verifying correctness. Because of the proliferation of logics of programming and the expense of theorem provers, it is useful to know when sentences in one logic can be translated into sentences in another logic in such a way that it is sound to apply a theorem prover for the second logic to the translated sentences.

This paper approaches these problems through the theory of institutions, where an **institution** is a logical system suitable for programming methodology; the paper also carries out some basic methodological work in an arbitrary institution. 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 set of Σ-sentences,
- a set of Σ-models, and
- a Σ-satisfaction relation, of Σ-sentences by Σ-models,

such that when you change signatures (with a signature morphism), the satisfaction relation between sentences and models changes consistently. One main result in this paper is that any institution whose syntax is nice enough to support gluing together interface declarations (as given by signatures) will also support gluing together **theories** (which are collections of sentences) to form larger specifications. A second main result shows how a suitable **institution morphism** permits a theorem prover for one institution to be used on theories from another. A third main result is that any institution supporting free constructions extends to another institution whose sentences may be either the old sentences, or else any of several kinds of constraint on interpretations. Such constraints are useful, for example, in data type declarations, and we believe that they are valuable as a general formulations of the kinds of induction used in computer science. Again using the notion of institution morphism, it is shown how sentences from one institution can be combined with constraints from another in a "duplex" institution; more generally, sentences and constraints from several institutions can be mixed together in a "multiplex" institution. This gives a very rich and flexible framework for program specification and other areas of theoretical computer science.

The notion of institution was first introduced as part of our research on Clear, in [Burstall & Goguen 80] under the name "language." The present paper adds many new concepts and results, as well as an improved notation. The "abstract model theory" of [Barwise 74] resembles our work in its intention to generalize basic results in model theory and in its use of elementary category theory; it differs in being more concrete (for example, its syntactic structures are limited to the usual function, relation and logical symbols) and in the results that are generalized (these are results of classical logic, such as those of Lowenheim-Skolem and Hanf).

The first two subsections below try to explain the first two paragraphs of this introduction a little more gradually. The third indicates what we will assume the reader knows about category theory. Section 2 is a brief review of general algebra, emphasizing results used to show that equational logic is indeed an institution. Section 3 introduces the basic definitions and results for institutions and theories, and considers the equational, first order, first order with equality, Horn clause, and conditional equational institutions. Section 4 discusses the use

of constraints to specify abstract data types. Section 5 considers the use of two or more institutions together. Section 6 discusses applications to programming methodology and shows how certain general specification concepts can be expressed in any suitable institution. All of the more difficult proofs and many of the more difficult definitions are omitted in this condensed version of the paper; these details will appear elsewhere at a later time.

1.1 Specifications and Logical Systems

Systematic program design requires careful specification of the problem to be solved. But recent experience in software engineering shows that there are major difficulties in producing consistent and rigorous specifications to 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 and re-usable pieces. We suggest that this modularity may be useful not only for understanding and writing specifications, but also for proving theorems about them, and in particular, for proving that a given program actually satisfies its specification. Modern work in programming methodology supports the view that abstractions, and in particular data abstractions, are a useful way to obtain such modularity, and that parameterized (sometimes called "generic") specifications dramatically enhance this utility. One way to achieve these advantages is to use a specification language that puts together parameterized abstractions.

It is important to distinguish between specifications that are written directly in some logical system, such as first order logic or the logic of some particular mechanical theorem prover, and specifications that are written in a genuine specification language, such as Special [Levitt, Robinson & Silverberg 79], Clear [Burstall & Goguen 77], OBJ [Goguen & Tardo 79], or Affirm [Gerhard, Musser et al. 79]. The essential purpose of a logical system is to provide a relationship of "satisfaction" between its *syntax* (i.e., its theories) and its *semantics* or models; this relationship is sometimes called a "model theory" and may appear in the form of a "Galois connection" (as in Section 3.2 below). A specification written directly in such a logical system is simply an unstructured, and possibly very large, set of sentences. On the other hand, the essential purpose of a specification language is to make it easy to write and to read specifications of particular systems, and especially of large systems. To this end, it should provide mechanisms for putting old and well-understood specifications together to form new specifications. In particular, it should provide for parameterized specifications and for constructions (like blocks) that define local environments in which specification variables may take on local values.

In order for a specification written in a given specification language to have a precise meaning, it is necessary for that language to have a precise semantics! Part of that semantics will be an underlying logic which must have certain properties in order to be useful for this task. These properties include a suitable notion of model; a satisfaction relationship between sentences and models; and a complete and reasonably simple proof theory. Examples include first order logic, equational logic, temporal logic, and higher order logic. In general, any mechanical theorem prover will have its own particular logical system (e.g., [Boyer & Moore 80], [Aubin 76], [Shostak, Schwartz & Melliar-Smith 81]) and the utility of the theorem prover will depend in part upon the appropriateness of that logical system to various application areas.

There is also the interesting possibility of using two (or even more) institutions together, as discussed in Section 5 below: then one can specify data structures in one institution and at the same time use the more powerful axioms available in other institutions; this has been used to advantage in some specifications in Clear [Burstall & Goguen 81] and also in the language

Ordinary [Goguen 82a, Goguen 82b]. This permits utilizing induction in institutions where it makes sense, and also writing sentences in other more expressive institutions where induction does not always make sense.

We wish to emphasize that a specification language is not a programming language. For example, the denotation of an Algol text is a function, but the denotation of a specification text is a **theory**, that is, a set of sentences *about* programs.

1.2 Parameterization over the Underlying Logic

Since a specification language is intended to provide mechanisms for structuring specifications, its definition and meaning should be as *independent* as possible of what underlying logical system is used. In fact, dependence on the logical system can be relegated to the level of the syntax of the sentences that occur in theories. This also serves to simplify the task of giving a semantics for a specification language.

In order to achieve the benefits listed above, we introduce the notion of an institution, which is an abstraction of the notion of a logical system [Burstall & Goguen 80]. We have designed our specification languages Clear and Ordinary in such a way that they can be used with *any* institution. This means, in particular, that they could be used in connection with any theorem prover whose underlying logic is actually an institution. Roughly speaking, an **institution** is a collection of signatures (which are vocabularies for constructing sentences) together with for each signature Σ: the set of all Σ-sentences; the category of all Σ-models; and a Σ-satisfaction relationship between sentences and models, such that when signatures are changed (with a signature morphism), satisfaction is preserved.

1.3 Prerequisites

Although relatively little category theory is required 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 that the reader is acquainted with the notions of category, functor and natural transformation. Occasional non-essential remarks use adjoint functors. There are several introductions to these ideas which the unfamiliar reader may consult, including [Arbib & Manes 75], [Goguen, Thatcher, Wagner & Wright 75a] and [Burstall & Goguen 82], and for the mathematically more sophisticated [MacLane 71] and [Goldblatt 79]. Familiarity with the initial algebra approach to abstract data types is helpful, but probably not necessary. Colimits are briefly explained in Section 3.4.

1.4 Acknowledgements

Thanks go to the Science Research Council of Great Britain for financial support and a Visiting Fellowship for JAG, to the National Science Foundation for travel money, to the Office of Naval Research and the National Science Foundation for research support, to Eleanor Kerse for typing some early drafts, and to Jose Meseguer for his extensive comments. We are grateful to many people for helpful conversations and suggestions, notably to our ADJ collaborators Jim Thatcher, Eric Wagner, and Jesse Wright, also to Peter Dybjer, Gordon Plotkin, David Rydeheard, Don Sanella, John Reynolds and Steve Zilles. Special thanks to Kathleen Goguen and Seija-Leena Burstall for extreme patience.

2 General Algebra

This section is quick review of many-sorted general algebra. This will provide our first example of the institution concept, and will also aid in working out the details of several other institutions. If you know all about general algebra, you can skip to Section 3, which defines the abstract notion of institution and gives some further examples. We will use the notational approach of [Goguen 74] (see also [Goguen, Thatcher & Wagner 78]) based on "indexed sets," in contrast to the more complex notations of [Higgins 63], [Benabou 68] and [Birkhoff & Lipson 70]. If I is a set (of "indices"), then an **I-indexed set** (or a family of sets indexed by I) A 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→B, is just an I-indexed family of functions f_i: A_i→B_i one for each i in I. There is an obvious composition[2] of I-indexed mappings, $(f;g)_i=f_i;g_i$ where f;g denotes composition of the functions f and g in the order given by (f;g)(x)=g(f(x)). This gives a category $\mathbf{Set}_I$ of I-indexed sets. We may use the notations $A=\langle A_i \mid i \text{ in } I\rangle$ for an I-indexed set with components A_i and $f=\langle f_i \mid i \text{ in } I\rangle$ for an I-indexed mapping A→B of I-indexed sets where f_i: $A_i \to B_i$. Notice that the basic concepts of set theory immediately extend component-wise to I-indexed sets. Thus, $A \subseteq B$ means that $A_i \subseteq B_i$ for each i in I, $A \cap B=\langle A_i \cap B_i \mid i \text{ in } I\rangle$, etc.

2.1 Equational Signatures

Intuitively speaking, an equational signature declares some sort symbols (to serve as names for the different kinds of data around) and some operator symbols (to serve as names for functions on these various kinds of data), where each operator declaration gives a tuple of input sorts, and one output sort. A morphism between signatures should map sorts to sorts, and operators to operators, so as to preserve their input and output sorts.

Definition 1: 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 $S^*\mathbf{x}S$; we will often write just Σ instead of $\langle S,\Sigma\rangle$. σ in Σ_{us} is said to have **arity** u, **sort** s, and **rank** u,s; we may write σ: u→s to indicate this. []

In the language of programming methodology, a signature declares the interface for a package, capsule, module, object, abstract machine, or abstract data type (unfortunately, there is no standard terminology for these concepts).

Definition 2: 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→S' of sorts and an $S\mathbf{x}S^*$-indexed family of maps g_{us}: $\Sigma_{us} \to \Sigma'_{f^*(u)f(s)}$ of operator symbols, where f*: S*→S'* is the extension of f to strings[3]. We will sometimes write $\phi(s)$ for f(s), $\phi(u)$ for f*(u), and $\phi(\sigma)$ or $\phi\sigma$ for $g_{us}(\sigma)$ when $\sigma \in \Sigma_{us}$. []

The signature morphism concept is useful for expressing the *binding* of an actual parameter to the formal parameter of a parameterized software module (see Section 6.2). As is standard in category theory, we can put the concepts of Definitions 1 and 2 together to get

Definition 3: The **category of equational signatures**, denoted **Sig**, has equational

---

[2] This paper uses ; for composition in any category.

[3] This extension is defined by: f*(λ)=λ, where λ denotes the empty string; and f*(us)=f*(u)f(s), for u in S* and s in S.

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.) []

2.2 Algebras

Intuitively, 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 methodology, correspond to concrete data types, i.e., to data representations in the sense of [Hoare 72].

Definition 4: Let $\langle S,\Sigma\rangle$ be a signature. Then a **Σ-algebra** A is an S-indexed family of sets $|A|=\langle A_s \mid s \text{ in } S\rangle$ called the **carriers** of A, together with an $S^*\mathbf{x}S$-indexed family α of maps α_{us}: $\Sigma_{us}\rightarrow[A_u\rightarrow A_s]$ for u in S^* and s in S, where $A_{s1...sn}=A_{s1}\mathbf{x}...\mathbf{x}A_{sn}$ and $[A\rightarrow B]$ denotes the set of all functions from A to B. (We may sometimes write for A for $|A|$ and A_s for $|A|_s$.) For $u=s1...sn$, for σ in Σ_{us} and for $\langle a1,...,an\rangle$ in A_u we will write $\sigma(a1,...,an)$ for $\alpha_{us}(\sigma)(a1,...,an)$ if there is no ambiguity. []

Definition 5: Given a signature $\langle S,\Sigma\rangle$ a **Σ-homomorphism** from a Σ-algebra $\langle A,\alpha\rangle$ to another $\langle A',\alpha'\rangle$, is an S-indexed map $f: A\rightarrow A'$ such that for all σ in Σ_{us} and all $a=\langle a1,...,an\rangle$ in A_u the **homomorphism condition**

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

holds. []

Definition 6: The **category $\mathbf{Alg}_\Sigma$ of Σ-algebras** has Σ-algebras as objects and Σ-homomorphism as morphisms; composition and identity are composition and identity as maps. (This clearly forms a category). []

We can extend **Alg** to a functor on the category **Sig** of signatures: it associates with each signature Σ the category of all Σ-algebras, and it also defines the effect of signature morphisms on algebras. In the definition below, $\mathbf{Cat}^{op}$ denotes the opposite of the category **Cat** of all categories, i.e., $\mathbf{Cat}^{op}$ is **Cat** with its morphisms reversed.

Definition 7: The functor **Alg**: $\mathbf{Sig}\rightarrow\mathbf{Cat}^{op}$ takes each signature Σ to the category of all Σ-algebras, and takes each signature morphism $\phi=\langle f: S\rightarrow S', g\rangle: \Sigma\rightarrow\Sigma'$ to the functor $\mathbf{Alg}(\phi)$: $\mathbf{Alg}_{\Sigma'}\rightarrow\mathbf{Alg}_\Sigma$ sending

1. 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. sending a Σ'-homomorphism $h': A'\rightarrow B'$ to the Σ-homomorphism $\mathbf{Alg}(\phi)(h')=h$: $\mathbf{Alg}(\phi)(A')\rightarrow\mathbf{Alg}(\phi)(B')$ 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 $\mathbf{Alg}(\phi)(h')$. []

If S is the sort set of Σ, then there is a **forgetful functor** U: $\mathbf{Alg}_\Sigma\rightarrow\mathbf{Set}_S$ sending each algebra to its S-indexed family of carriers, and sending each Σ-homomorphism to its underlying S-indexed map.

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_Σ forms a Σ-algebra in a natural way.

We begin our more precise discussion with a 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. σ in $\Sigma_{s1...sn,s}$ and ti in $T_{\Sigma,si}$ implies the string $\sigma(t1,...,tn)$ is in $T_{\Sigma,s}$.

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

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

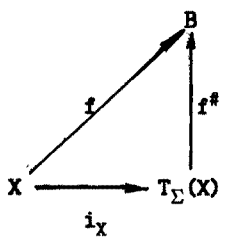
Next, we define $\Sigma(X)$ to be the S-sorted signature with $(\Sigma(X))_{us}=\Sigma_{us}\cup X_s$ if $u=\lambda$ and $(\Sigma(X))_{us}=\Sigma_{us}$ if $u\neq\lambda$. Then $T_\Sigma(X)$ is just $T_{\Sigma(X)}$ regarded as a Σ-algebra rather than as a $\Sigma(X)$-algebra.

The freeness of $T_\Sigma(X)$ is expressed precisely by the following.

Theorem 8: Let i_X: $X\rightarrow U(T_\Sigma(X))$ denote the inclusion described above. Then the following "universal" property holds: for any Σ-algebra B, every (S-indexed) map f: $X\rightarrow B$, called an **assignment**, extends uniquely to a Σ-homomorphism $f^\#$: $T_\Sigma(X)\rightarrow B$ such that $i_X;U(f^\#)=U(f)$.

[]

We will often omit the U's in such equations, as in the following traditional diagram of S-indexed sets and mappings for the above equation:



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

2.3 Equations and Satisfaction

We now define the equations over a given signature, and what it means for an algebra to satisfy an equation.

Definition 9: A **Σ-equation** e is a triple $(X,\tau1,\tau2)$ where X is an S-indexed set (of variable symbols) and $\tau1$ and $\tau2$ in $|T_\Sigma(X)|_s$ are terms over X of the same sort s in S. Such an equation might be written

for all X, $\tau1=\tau2$

or

$(\forall X)\ \tau1=\tau2$.

[]

The necessity for explicitly including declarations for variables in equations (as in [Burstall & Goguen 77]) is shown in [Goguen & Meseguer 81]: without this one gets an unsound deductive system for many-sorted equational logic.

Definition 10: A Σ-algebra A **satisfies** a Σ-equation $(X,\tau 1,\tau 2)$ iff for all assignments f: X→|A| we have $f^{\#}(\tau 1)=f^{\#}(\tau 2)$. We will write A |= e for "A satisfies e." []

We now define another functor, Eqn, on the category of signatures. In order to do so, we first define for each signature morphism ϕ: $\Sigma\rightarrow\Sigma'$ a function $\phi^{\sim}$ from Σ-terms to Σ'-terms. To this end, we first give a simpler definition for sort maps.

Definition 11: If X is an S-sorted set of variables and if f: S→S', then we define $f^{\sim}(X)$ to be the S'-indexed set X' such that

$$X'_{s'}=\cup\{X_s \mid f(s)=s'\} .$$

(Notice that without loss of generality, we can assume disjointness of the sets X_s for s in S: if the X_s were not disjoint, we could just take a disjoint union in the above formula.)

Now let ϕ: $\Sigma\rightarrow\Sigma'$ be the signature morphism $\langle$f: S→S', g$\rangle$. Let X be an S-indexed set (of variables) and let X' be $f^{\sim}(X)$. We will define an S-indexed map $\phi^{\sim}$: $|T_{\Sigma}(X)|\rightarrow|\phi(T_{\Sigma'}(X'))|$. First, note that[4] $X\subseteq|\phi(T_{\Sigma'}(X'))|$ since if x is in X_s then x is in $X'_{f(s)}$ and $X'_{f(s)}\subseteq|T_{\Sigma'}(X')|_{f(s)}=|\phi(T_{\Sigma'}(X'))|_s$; let j: X→$|\phi(T_{\Sigma'}(X'))|$ denote this inclusion. Then j has a unique extension as a Σ-homomorphism $j^{\#}$: $T_{\Sigma}(X)\rightarrow\phi(T_{\Sigma'}(X'))$ by Theorem 8, and we simply define $\phi^{\sim}=|j^{\#}|$. []

Definition 12: The functor Eqn: **Sig→Set** takes each signature Σ to the set Eqn(Σ) of all Σ-equations, and takes each $\phi=\langle f,g\rangle$: $\Sigma\rightarrow\Sigma'$ to the function Eqn(ϕ): Eqn(Σ)→Eqn(Σ') defined by

$$Eqn(\phi)(\langle X,\tau 1,\tau 2\rangle)=\langle f^{\sim}(X),\phi^{\sim}(\tau 1),\phi^{\sim}(\tau 2)\rangle .$$

It is often convenient to write ϕ(e) or ϕe instead of Eqn(ϕ)(e). []

Proposition 13: Satisfaction Condition. If ϕ: $\Sigma\rightarrow\Sigma'$, if e is an Σ-equation, and if A' is a Σ'-algebra, then

$$A' \models \phi(e) \text{ iff } \phi(A') \models e . \quad []$$

The not entirely trivial proof is omitted from this version of the paper. This concludes our review of general algebra. We now turn to our generalization.

3 Institutions

An institution consists of a category of signatures such that associated with each signature are sentences (e.g., equations), models (e.g., algebras), and a relationship of satisfaction that is, in a certain sense, invariant under change of signature. This is an abstraction of first order model theory. A different approach, axiomatizing the category of theories as well as that of signatures, is given in [Goguen & Burstall 78]; another uses the notion of monadic theory [Burstall & Rydeheard 80]. The generality of these approaches is useful in dealing with aspects of programming, such as errors and infinite data structures, that seem to require theories and models that are in some way more complicated than those of ordinary logic. Another motivation is the elegant way that data definitions can be handled (see Section 5).

---

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

3.1 Definition and Examples

The essence of the notion of institution is that when signatures are changed (with a signature morphism) then sentences and models change consistently. This consistency is expressed by the "Satisfaction Condition," which goes a step beyond the classical conception of "semantic truth" in [Tarski 44]. The wide range of consequences and the fact that Proposition 13 is not entirely trivial, suggest that this is not a trivial step. A philosophical argument can also be given: it is a familiar and basic fact that the truth of a sentence (in logic) is independent of the vocabulary chosen to represent the basic relationships that occur in it. It is also fundamental that sentences translate in the same direction as a change of symbols, while models translate in the opposite direction; the Satisfaction Condition is an elegant expression of the invariance of truth under the renaming of basic symbols taking account of the variance of sentences and the covariance of models. (It also generalizes a condition called the "Translation Axiom" in [Barwise 74].)

Definition 14: An **institution** I consists of

1. a category **Sign** of "signatures,"
2. a functor Sen: **Sign**$\rightarrow$**Set** giving the set of **sentences** over a given signature,
3. a functor **Mod**: **Sign**$\rightarrow$**Cat**^{op} giving the category (sometimes called the **variety**) of **models** of a given signature (the arrows in **Mod**(Σ) are called **model morphisms**), and
4. a **satisfaction** relation $\models \subseteq |\mathbf{Mod}(\Sigma)| \times \mathrm{Sen}(\Sigma)$ for each Σ in **Sign**, sometimes denoted $\models_\Sigma$

such that for each morphism ϕ: $\Sigma \rightarrow \Sigma'$ in **Sign**, the **Satisfaction Condition**

$$m' \models \phi(e) \text{ iff } \phi(m') \models e$$

holds for each m' in $|\mathbf{Mod}(\Sigma')|$ and each e in $\mathrm{Sen}(\Sigma)$. []

For some purposes this definition can be simplified by replacing **Mod**: **Sign**$\rightarrow$**Cat**^{op} by Mod: **Sign**$\rightarrow$**Set**^{op}. Then Mod(Σ) is the *set* of all Σ-models; the two versions of the definition are thus related by the equation Mod(Σ)=$|\mathbf{Mod}(\Sigma)|$. Indeed, our original version [Burstall & Goguen 80] was the second. Some reasons for changing it are: first, it is more consistent with the categiorical point of view to consider morphisms of models along with models; and secondly, we would like every liberal institution to be an institution, rather than just to determine one (liberal institutions will play an important role later in this paper).

There is a more categorical definition of the institution concept, replacing the rather ad hoc looking family of satisfaction relations by a functor into a category of "twisted relations." This will be given in the full version of this paper.

Example 1: Equational Logic. The work of Section 2 shows that (many sorted) equational logic is an institution, with **Sign** the category **Sig** of equational signatures (Definition 3), with **Mod**(Σ)=**Alg**(Σ) (Definition 6; see Definition 7 for the functor **Alg** which instantiates the functor **Mod**), with Sen(Σ) the set of all Σ-equations (Definition 8; see Definition 12 for the functor Eqn, which instantiates the functor Sen), and with satisfaction given in the usual way (Definition 10). The Satisfaction Condition holds by Proposition 13. Let us denote this institution $\mathcal{EQ}$. **End of Example 1.**

Example 2: First Order Logic. We follow the path of Section 2 again, but now showing that a more complicated logical system is an institution. Our work on equational logic will greatly aid with this task. Many details are omitted.

Definition 15: 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*xS-indexed family of sets (of **operator** symbols, also called **function** symbols), and
3. Π is an S*-indexed family of sets (of **predicate symbols**).

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

1. ϕ_1: S→S' is a function,
2. ϕ_2: $\Sigma\to\Sigma'$ is an S*xS-indexed family of functions $(\phi_2)_{us}$: $\Sigma_{us}\to\Sigma'_{\phi_1^*(u)\phi_1(s)}$ and
3. ϕ_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 and with first order signature morphisms as its morphisms. []

Definition 16: 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 $\langle A_s \mid s \text{ in } S\rangle$, where A_s is called the **carrier** of sort s,
2. an S*xS-indexed family α of functions α_{us}: $\Sigma_{us}\to[A_u\to A_s]$ assigning a function to each function symbol, and
3. an S*-indexed family β of functions β_u: $\Pi_u\to Pow(A_u)$ assigning a relation to each predicate symbol, where Pow(X) denotes the set of all subsets of a set X.

For π in Π_u with u=s1...sn and ai in A_{si} for i=1,...,n, we say that "π(a1,...,an) holds" iff (a1,...,an) is in $\beta(\pi)$; and as usual, we may abbreviate this assertion by simply writing "π(a1,...,an)."

Next, we define a **first order Ω-homomorphism** f: A→A' of Ω-models A and A' 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 π in Π_u with u=s1...sn, and with ai in A_{si} for i=1,...,n,

π(a1,...,an) implies $\pi'(f_{s1}(a1),...,f_{sn}(an))$,

where π denotes $\beta(\pi)$ and π' denotes $\beta'(\pi)$.

Let **FoMod** denote the category with first order models as its objects and with first order morphisms as its morphisms. We now extend **FoMod** to a functor **FoSig**→**Cat**^{op}. Given a first order signature morphism ϕ: $\Omega\to\Omega'$, define the functor **FoMod**(Ω')→**FoMod**(Ω) to send: first of all, A' in **FoMod**(Ω') to A=ϕA' defined by

1. $A_s=A'_{s'}$ for s in S with s'=ϕ_1(s),
2. $\alpha_{us}(\sigma)=\alpha'_{u's'}((\phi_2)_{us}(\sigma))$ for u in S*, s in S and σ in Σ_{us} where u'=ϕ_1*(u) and s'=ϕ_1(s), and
3. $\beta_{us}(\pi)=\beta'_{u'}((\phi_3)_u(\pi))$ for u in S* and π in Π_u with u' as above;

and secondly, to send f': A'→B' in **FoMod**(Ω') to f=ϕf': A→B in **FoMod**(Ω), where A=ϕA' and B=ϕB', defined by $f_s=f'_{s'}$ where s'=ϕ_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 formulas. Let X be an S-indexed set of variable symbols, with each X_s the infinite set $\{x_1^s, x_2^s, ...\}$; we may omit the superscript s if the sort is clear from context. Now define the S-indexed family TERM(Ω) of **Ω-terms** to be the carriers of $T_\Sigma(X)$, the free Σ-algebra with generators X, and define a(n S-indexed) function Free on TERM(Ω) inductively by

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

Definition 17: A (well formed) **Ω-formula** is an element of the carrier of the (one sorted) free algebra WWF(Ω) having the **atomic formulae** $\{ \pi(t1,...,tn) \mid \pi \in \Pi_u$ with u=s1...sn and ti $\in$ $TERM(\Omega)_{si} \}$ as generators, and having its (one sorted) signature composed of

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.

The functions Var and Free, giving the sets of **variables** and of **free variables** that are used, respectively, in Ω-formulae, can be defined in the usual way, inductively over the above logical connectives. We then define Bound(P)=Var(P)-Free(P), the set of **bound variables** of P.

We can now define the remaining logical connectives in terms of the basic ones given above in the usual way.

Finally, define an **Ω-sentence** to be a **closed** Ω-formula, that is, an Ω-formula P with $Free(P)=\emptyset$. Finally, let FoSen(Ω) denote the set of all Ω-sentences. []

We now define the effect of FoSen on first order signature morphisms, so that it becomes a functor **FoSig$\rightarrow$Set**. Given ϕ: $\Omega \rightarrow \Omega'$, we will define FoSen($\phi$): FoSen($\Omega$)$\rightarrow$FoSen($\Omega'$) using the initiality of TERM(Ω) and WFF(Ω). Since (ϕ_1,ϕ_2): $\Sigma \rightarrow \Sigma'$ is a signature morphism, there is an induced morphism ψ: $T_\Sigma(X) \rightarrow T_{\Sigma'}(X)$ which then gives ψ: TERM(Ω)$\rightarrow$TERM(Ω'). We can now define WFF(ϕ): WFF(Ω)$\rightarrow$WFF(Ω') by its effect on the generators of WFF(Ω), which are the atomic formulae, namely

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

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

It remains to define satisfaction. This corresponds to the usual "semantic definition of truth" (originally due to [Tarski 44]) and is again defined inductively. If A is a first order model, let Asgn(A) denote the set of all **assignments** of values in A to variables in X, i.e, [X$\rightarrow$A], the set of all S-indexed functions f: X$\rightarrow$A.

Definition 18: Given a sentence P, define Asgn(A,P), the set of assignments in A for which P is true, inductively by

1. if $P=\pi(t1,...,tn)$ then $f \in Asgn(A,P)$ 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, using the values of variables given by the assignment f,

2. Asgn(A,true)=Asgn(A),
3. Asgn(A,¬P)=Asgn(A)-Asgn(A,P),
4. Asgn(A,P&Q)=Asgn(A,P)∩Asgn(A,Q), and
5. Asgn(A,(∀x)P)={f | Asgn(A,f,x)⊆Asgn(A,P)}, where Asgn(A,f,x) is the set of all assignments f′ that agree with f except possibly on the variable x.

Then a model A **satisfies** a sentence P, written A |= P, iff Asgn(A,P)=Asgn(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. Thus, first order logic is an institution; let us denote it $\mathcal{FO}$. **End of Example 2.**

Example 3: First Order Logic with Equality. This institution is closely related to that of Example 2. A signature for first order logic with equality is a first order signature $\Omega=(S,\Sigma,\Pi)$ 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 must preserve these predicate symbols, i.e., $\phi_3(\equiv_s)=\equiv_{\phi_1(s)}$. 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 for all s in S, and for all a,a′ in A_s, $a\equiv_s a'$ iff a=a′.

A homomorphism of first order Ω-models with equality is just a first order Ω-homomorphism (in the sense of Definition 16), and we get a category **FoModEq**(Ω) of Ω-models for each signature Ω in |**FoSigEq**|, and **FoModEq** is a functor on **FoSigEq**. Ω-sentences are defined just as in Example 2, and so is satisfaction. We thus get a functor FoSenEq: **FoSigEq→Set**. The Satisfaction Condition follows immediately from that of first order logic without equality. Let us denote this institution by $\mathcal{FOEQ}$. **End of Example 3.**

Example 4: Horn Clause Logic with Equality. We now specialize the previous example by limiting the form that sentences can take, but without resticting either the predicate or operator symbols that may enter into them. In particular, we maintain the equality symbol with its fixed interpretation from Example 3; but we require that all sentences be of the form

$$(\forall \underline{x})\ A_1\&A_2\&\ldots A_n \Rightarrow A\ ,$$

where each A_i is an atomic formula $\pi(t_1,\ldots,t_m)$. In particular, we do not allow disjunction, negation or existential quantifiers. That this is an institution follows from the fact that first order logic with equality is an institution. Let us denote this institution $\mathcal{HORN}$. **End of Example 4.**

Example 5: Conditional Equational Logic. As a specialization of the first order Horn clause logic with equality, we can consider the case where equality is the only predicate symbol. This gives the institution often called conditional equational logic. **End of Example 5.**

Example 6: Horn Clause Logic without Equality. We can also restrict Example 4 by dropping equality with its fixed interpretation. This too is obviously an institution because it is just a restriction of ordinary first order logic. **End of Example 6.**

It seems clear that we can do many-sorted temporal or modal logic in much the same way, by adding the appropriate modal operators to the signature and defining their correct interpretation in all models; the models may be "Kripke" or "alternative world" structures. Higher order equational logic is also presumably an institution; the development should again

follow that of equational logic, but using higher order sorts and operator symbols.[5] Quite possibly, the "inequational logic" of [Bloom 76], the order-sorted equational logic of [Goguen 78], and various kinds of infinitary equational logic, such as the logic of continuous algebras in [Goguen, Thatcher, Wagner & Wright 75b] and [Wright, Thatcher, Wagner & Goguen 76] are also institutions. We further conjecture that in general, mechanical theorem provers, such as those of [Boyer & Moore 80], [Aubin 76] and [Shostak, Schwartz & Melliar-Smith 81], are based on logical systems that are institutions (or if they are not, should be modified so that they are!). Clearly, it would be helpful to have some general results to help in establishing whether or not various logical systems are institutions.

There is a way of generating many other examples due to [Mahr & Makowsky 82a]: take as the sentences over a signature Σ all specifications using Σ written in some specification language (such as Special or Affirm); we might think of such a specification as a convenient abbreviation for a (possibly very large) conjunction of simpler sentences. However, this seems an inappropriate viewpoint in the light of Section 6 which argues that the real purpose of a specification language is to define the meanings of the basic symbols (in Σ), so that the signature should be constructed (in a highly structured manner) right along with the sentences, rather than being given in advance.

This and the next section assume that a fixed but arbitrary institution has been given; Section 5 discusses what can be done with more than one institution.

3.2 Theories and Theory Morphisms

A Σ-theory, often called just a "theory" in the following, consists of a signature Σ and a closed set of Σ-sentences. Thus, this notion differs from the [Lawvere 63] notion of "algebraic theory," which is independent of any choice of signature; the notion also simplifies the "signed theories" of [Burstall & Goguen 77]. The simpler notion is more appropriate for the purposes of this paper, as well as easier to deal with. In general, our theories contain an infinite number of sentences, but are defined by a finite presentation.

Definition 19:

1. A **Σ-theory presentation** is a pair (Σ,E), where Σ is a signature and E is a set of Σ-sentences.
2. A Σ-model A **satisfies** a theory presentation (Σ,E) if A satisfies each sentence in E; let us write $A \models E$.
3. If E is a set of Σ-sentences, let E^* be the set[6] of all Σ-models that satisfy each sentence in E.
4. If M is a set of Σ-models, let M^* be the set of all Σ-sentences that are satisfied by each model in M; we will hereafter also let M^* denote (Σ,M^*), the **theory of** M.
5. By the **closure** of a set E of Σ-sentences we mean the set E^{**}, written $E^\bullet$.
6. A set E of Σ-sentences is **closed** iff $E=E^\bullet$. Then a **Σ-theory** is a theory presentation (Σ,E) such that E is closed.

---

[5] The sorts involved will be the objects of the free Cartesian closed category on the basic sort set [Parsaye-Ghomi 82].

[6] In the terminology of axiomatic set theory, this will often turn out to be a **class** rather than a **set** of models, but this distinction will be ignored in this paper.

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

[]

Notice that we have given a model-theoretic definition of closure. For some institutions, a corresponding proof-theoretic notion can be given, because there is a complete set of inference rules. For the equational institution, these rules embody the equivalence properties of equality, including the substitution of equal terms into equal terms [Goguen & Meseguer 81].

We can also consider closed sets of models (in the equational institution these are usually called varieties). The **closure** of a set M of models is M^{**}, denoted $M^\bullet$, and a full subcategory of models is called **closed** iff its objects are all the models of some set of sentences.

Definition 20: 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 $F: \Sigma\to\Sigma'$ such that $\phi(e)$ is in E′ for each e in E; we will write $F: T\to T'$. The **category of theories** has theories as objects and theory morphisms as morphisms, with their composition and identities defined as for signature morphisms; let us denote it **Th**. (It is easy to see that this is a category.) []

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

Proposition 21: The two functions, *: sets of Σ-sentences $\to$ sets of Σ-models, and *: sets of Σ-models $\to$ sets of Σ-sentences, given in Definition 19, form what is known as a **Galois connection** [Cohn 65], in that they satisfy the following properties, for any sets E,E′ of Σ-sentences and sets 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 sets of sentences and the closed sets of models.
8. $(\cup_n E_n)^*=\cap E_n{}^*$.
9. $\phi(E^*)=(\phi E)^*$, for $\phi: \Sigma\to\Sigma'$ a signature morphism.

[]

For a theory T with signature Σ, let **Mod**(T) and also T^* denote the **full subcategory** of **Mod**(Σ) of all Σ-models that satisfy all the sentences in T. This is used in the following.

Definition 22: Given a theory morphism $F: T\to T'$, the **forgetful functor** $F^*: T'^*\to T^*$ sends a T′-model m′ to the T-model F(m′), and sends a T′-model morphism $f: m'\to n'$ to $F^*(f)=$**Mod**(F)(f): $F(m')\to F(n')$. This functor may also be denoted **Mod**(F). []

For this definition to make sense, we must show 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 us that $\phi(m')$

$\models$ e, as desired. We also need that the morphism $\phi^*(f)$ lies in T*, but this follows because T* is a full subcategory of **Mod**(Σ), and the source and target objects of $\phi^*(f)$ lie in T*.

3.3 The Closure and Presentation Lemmas

Let us write $\phi(E)$ for $\{\phi(e) \mid e \text{ is in } E\}$ and $\phi(M)$ for $\{\phi(m) \mid m \text{ is in } M\}$. Let us also write $\phi^{-1}(M)$ for $\{m \mid \phi(m) \text{ is in } M\}$. Using this notation, we can more compactly write the Satisfaction Condition as

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

and using this notation, we can derive

Lemma 23: 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 21. Therefore $\phi(E^\bullet)=\phi(E^{**}) \subseteq \phi(E^{**})^{**}=\phi(E)^{**}=\phi(E)^\bullet$, using 3. of Proposition 21 and the just proved equation. []

The following gives a necessary and sufficient condition for a signature morphism to be a theory morphism.

Lemma 24: Presentation. Let ϕ: $\Sigma \to \Sigma'$ and suppose that $\langle\Sigma,E\rangle$ and $\langle\Sigma',E'\rangle$ are presentations. Then ϕ: $\langle\Sigma,E^\bullet\rangle \to \langle\Sigma',E'^\bullet\rangle$ is a theory morphism iff $\phi(E) \subseteq E'^\bullet$.

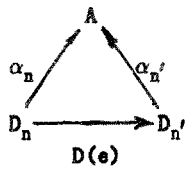
Proof: By the Closure Lemma, $\phi(E^\bullet) \subseteq \phi(E)^\bullet$. By hypothesis, $\phi(E) \subseteq E'^\bullet$. Therefore, $\phi(E^\bullet) \subseteq \phi(E)^\bullet \subseteq E'^{\bullet\bullet}=E'^\bullet$, so ϕ is a theory morphism. Conversely, if ϕ is a theory morphism, then $\phi(E^\bullet) \subseteq E'^\bullet$. Therefore $\phi(E) \subseteq E'^\bullet$, since $E \subseteq E^\bullet$. []

If an institution has a complete set of rules of deduction, then the Presentation Lemma tells us that to check if ϕ is a theory morphism, we can apply ϕ to each sentence e of the source presentation E and see whether $\phi(e)$ can be proved from E'. There is no need to check all the sentences in $E^\bullet$.

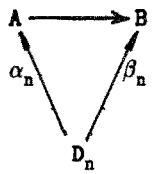
3.4 Putting Theories Together

A useful general principle is to describe a large widget as the interconnection of a system of small widgets, using widget-morphisms to indicate the interfaces over which the interconnection is to be done. [Goguen 71] (see also [Goguen & Ginali 78]) gives a very general formulation in which such interconnections are calculated as colimits. One application is to construct large theories as colimits of small theories and theory morphisms [Burstall & Goguen 77]. In particular, the pushout construction for applying parameterized specifications that was implicit in [Burstall & Goguen 77] has been given explicitly by [Burstall & Goguen 78] and [Ehrich 78]. For this to make sense, the category of theories should have finite colimits. For the equational institution, [Goguen & Burstall 78] have proved that the intuitively correct syntactic pasting together of theory presentations exactly corresponds to theory colimits. Colimits have also been used for many other things in computer science, for example, pasting together graphs in the theory of graph grammars [Ehrig, Kreowski, Rosen & Winkowski 78]. Let us now review the necessary categorical concepts.

Definition 25: A **diagram** D in a category **C** consists of a graph G together with a labelling of each node n of G by an object D_n of **C**, and a labelling of each edge e (from node n to node n' in G) by a morphism D(e) in **C** from D_n to $D_{n'}$; let us write D: G$\to$**C**. Then a **cone** α in **C** over the diagram D consists of an object A of **C** and a family of morphisms α_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 **C**. We call D the **base** of the cone α, A its **apex**, G its **shape**, and we write α: D⇒A. If α and β are cones with base D and apexes A,B (respectively), then a **morphism of cones** $\alpha \rightarrow \beta$ is a morphism f: A→B in **C** such that for each node n in G the diagram



commutes in **C**. Now let **Cone**(D,**C**) denote the resulting category of all cones over D in **C**. Then a **colimit** of D in **C** is an initial object in **Cone**(D,**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 the **colimit object**, and the morphisms α(n) to the apex are called the **injections** of D(n) into the colimit. The colimit object is also unique up to isomorphism.

Definition 26: 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 not proper classes). A functor F: **C**→**C'** **reflects colimits** iff whenever D is a diagram in **C** such that the diagram D;F in **C'** has a colimit cone α': D⇒A' in **C'**, then there is a colimit cone α: D⇒A in **C** with $\alpha' = \alpha;F$ (i.e., $\alpha'_n = F(\alpha_n)$ for all nodes n in the base of D). []

Here is the general result about putting together theories in any institution with a suitable category of signatures. The proof is omitted.

Theorem 27: The forgetful functor Sign: **Th**→**Sign** reflects colimits. []

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.

The category of equational signatures is finitely cocomplete (see [Goguen & Burstall 78] for a simple proof using comma categories), so we conclude that the category of signed equational theories is cocomplete. Using similar techniques, we can show that the category of first order signatures is cocomplete, and thus without effort conclude that the category of first order theories is cocomplete (this might even be a new result).

4 Constraints

To avoid overspecifying problems, we often want *loose* specifications, i.e., specifications that have many acceptable models. On the other hand, there are also many specifications where one wants to use the natural numbers, truth values, or some other fixed data type. In such cases, one wants the subtheories that correspond to these data types to be given *standard* interpretations. Moreover, we sometimes want to consider parameterized standard data types, such as **Set**[**X**] and **List**[**X**]. Here, one wants sets and lists to be given standard interpretations, once a suitable interpretation has been given for **X**.

So far we have considered the category T*=**Mod**(T) of *all* interpretations of a theory T; this section considers 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 constraints are the "data constraints" of [Burstall & Goguen 80], generalizing and relativizing the "initial algebra" approach to abstract data types introduced by [Goguen, Thatcher, Wagner & Wright 75b, Goguen, Thatcher & Wagner 78]. However, the first work in this direction seems to be that of [Kaphengst & Reichel 71], later generalized to "initially restricting algebraic theories" [Reichel 80]. Data constraints make sense for any "liberal" institution, and are more expressive even in the equational institution. Actually, our general results apply to many different notions of what it means for a model to satisfy a constraint. In particular, we will see that "generating constraints" and "hierarchy constraints" are special cases; these correspond to the conditions of "no junk" and "no confusion" that together give the more powerful notion of "data constraint." Section 5 generalizes this to consider "duplex constraints" that involve a different insititution than the one in which models are taken.

4.1 Free Interpretations

For example, suppose that we want to define the natural numbers in the equational institution. To this end, consider a theory **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 there is *exactly* one Σ-homomorphism from it 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. This characterization of the natural numbers is due to [Lawvere 64], and a proof that it is equivalent to Peano's axioms can be found in [MacLane & Birkhoff 67], pages 67-70.

The initial algebra approach to abstract data types [Goguen, Thatcher, Wagner & Wright 75b, Goguen, Thatcher & Wagner 78] has taken this "Lawvere-Peano" characterization of the natural numbers as a paradigm for defining other abstract data types; the method has been used for sets, lists, stacks, and many many other data types, and has even been used to specify database systems [Goguen & Tardo 79] and programming languages [Goguen & Parsaye-Ghomi 81]. 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 defined uniquely up to isomorphism. A number of less abstract equivalents of initiality for the equational institution, including a generalized Peano axiom system, are given in [Goguen & Meseguer 83].

Let us now consider the case of the parameterized abstract data type of sets of elements of a sort **s**. We add to **s** a new sort **set**, and operators[7]

$\emptyset$: **set**
$\{\_\}$: **s** $\rightarrow$ **set**
$\_\cup\_$: **set**,**set** $\rightarrow$ **set** ,

subject to the following equations, where S, S′ and S″ are variables of sort **set**,

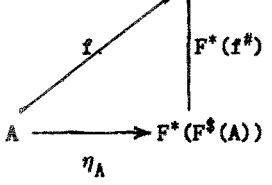
$\emptyset \cup S = S = S \cup \emptyset$
$S \cup (S' \cup S'') = (S \cup S') \cup S''$
$S \cup S' = S' \cup S$

[7] We use "mixfix" declarations in this signature, in the style of OBJ [Goguen 77, Goguen & Tardo 79]: the underbars are placeholders for elements of a corresponding sort from the sort list following the colon.

$S \cup S = S$.

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** (since there are no operators to generate elements of **s**) and has only the element $\emptyset$ of sort **set**. What we want is to permit *any* interpretation for the parameter sort **s**, and then to require that the new sort **set** and its new operators are interpreted freely *relative* to the given interpretation of **s**.

Let us make this precise. Suppose that $F: T \to T'$ is a theory morphism. Then there is a forgetful functor from the category of T'-models to the category of T-models, $F^*: T'^* \to T^*$ as in Definition 22. In the equational case $T^* = \mathbf{Alg}(T)$ and a very general result of [Lawvere 63] says that there is a functor that we will denote by $F^\$: T^* \to T'^*$ called the **free functor** determined by F, characterized by the following "universal" property: given a T-model A, there is a T'-model $F^\$(A)$ and a T-morphism $\eta_A: A \to F^*(F^\$(A))$, called the **universal** morphism, such that for any T'-model B' and any T-morphism $f: A \to F^*(B')$, there is a unique T'-morphism $f^\#: F^\$(A) \to B'$ such that the diagram



commutes in **Mod**(T). It can be shown [MacLane 71] that if for each A there is an object $F^\$(A)$ with this universal property, then there is a unique way to define $F^\$$ on morphisms so as to get a functor.

For this to make sense in an arbitrary institution, we need the existence of free constructions over arbitrary theory morphisms, as expressed in the following.

Definition 28: An institution is **liberal** iff for every theory morphism $F: T \to T'$ and every T-model A, there is a T'-model $F^\$(A)$ with a **universal** T-morphism $\eta_A: A \to F^*(F^\$(A))$ having the property that for each T'-model B' and T-morphism $f: A \to F^*(B')$, there is a unique T'-morphism $f^\#: F^\$(A) \to B'$ such that the above diagram commutes (in the category **Mod**(T)). []

As in the equational case, the existence of a universal morphism for each T-model A guarantees the existence of a unique functor $F^\$$ having the value $F^\$(A)$ on the object A [MacLane 71]; this functor $F^\$$ is called a **free** functor, just as F^* is called a **forgetful** functor. ($T^\$$ is left adjoint to F^*, and is unique up to isomorphism of its value objects if it exists. Thus an institution is liberal iff the forgetful functors induced by theory morphisms always have left adjoints.) The equational institution is liberal, but the first order logic instituion is not. The Horn clause first order logic with equality discussed in Example 4 is another liberal institution, by deep results of [Gabriel & Ulmer 71]; so is the conditional equational institution of Example 5.

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-of-Triv**, obtained by adding the sort **set** and the operators and equations given above. Then Set* takes a **Set-of-Triv**-algebra (which is just a set) and forgets the new sort **set** and the three new operators, giving an algebra that has just the set of **s**-sorted elements. The free functor $Set^\$$

takes a **Triv**-algebra A and extends it freely to a **Set-of-Triv**-algebra, the new operators giving distinct results except where the equations of **Set-of-Triv** force equality.

Given a **Set-of-Triv**-algebra B, there is a natural way to check whether or not its **set** sort and operators are free over its parameter sort **s**: in the above diagram, let A=Set*(B) and let $f=id_{Set^*(B)}$; then $f^\#$: $Set^\$(Set^*(B))\rightarrow B$ restricted to the parameter sort is the identity (because $Set^*(f^\#)=f$), and $f^\#$ itself should be an isomorophism if the **set** part of B is to be free over its parameter part. In the general case, the morphism $(id_{F^*(B)})^\#$: $F^\$(F^*(B))\rightarrow B$ is called the **counit** (of the adjunction) and denoted ϵ_B. The discussion of this example motivates the following.

Definition 29: Let F: T→T' be a theory morphism. Then a T'-model B is **F-free** iff the counit morphism $\epsilon_B=(id_{F^*(B)})^\#$: $F^\$(F^*(B))\rightarrow B$ is an isomorphism. []

The notion of F-free for the equational case is due to [Thatcher, Wagner & Wright 79][8] .

Results of [Mahr & Makowsky 82b] and [Mahr & Makowsky 82a] show that, in a sense, the most general sublanguage of first order logic that admits initial models is Horn clause logic with infinitary clauses; further, the most general finitary language uses finitary Horn clauses; the most general equational language uses (infinitary) conditional equations; and the most general finitary equational sublanguage consists of finitary conditional equations. These results apply to the existence of initial models, rather than to left adjoints of forgetful functors; it would be interesting to know whether or not they extend in this way. It is also interesting to note that they have made use of abstractions of the notion of "logical system" similar to that of an "institution."

4.2 Constraining Theories

It is very convenient in program specification to let theories include not just the sentences provided by an institution, but also declarations that certain subtheories should be interpreted freely relative others in the sense of Section 4.1. We call such sentences **data constraints** and we call theories that can include them **constraining theories**. Such a theory could require that subtheories defining data types such as natural numbers, sets, and strings be given their standard interpretations (uniquely up to isomorphism), while permiting a sort with a partial ordering to have any interpretation (that satisfies the partial order theory), and thus to be "loose," or to be parametric as are the "meta-sort" or "requirement" theories of Clear [Burstall & Goguen 77].

Returning to the Set example of the previous subsection, let us further enrich **Set-of-Triv** with some new sorts and operators to get a theory **Set-of-Triv-Etc**. For example, we might add an operator

choice: **set** → **s**

that chooses an arbitrary element of any non-empty set; this is a good example of a loose specification. For example, the enriched theory might include the conditional equation

choose(s) in s if $s\neq\emptyset$.

Let Etc: **Set-of-Triv**→**Set-of-Triv-Etc** be the theory inclusion and let

etc: Sign(**Set-of-Triv**)→Sign(**Set-of-Triv-Etc**)

---

[8] [Burstall & Goguen 80] defined B to be F-free if $B\approx F^\$(F^*(B))$; however, there are examples in which these two objects are isomorphic, but not naturally so by the counit morphism η. Our thanks to Eric Wagner for this comment.

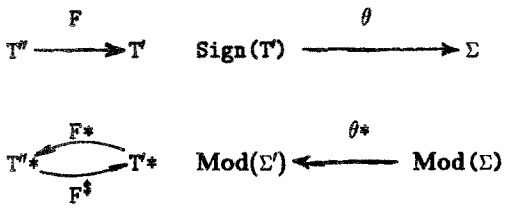
be the corresponding signature morphism. Then a **Set-of-Triv-Etc**-algebra A interprets sets as intended if etc*(A) satisfies **Set-of-Triv** and is Set-free. This motivates the following.

Definition 30: Let Σ be a signature. Then a Σ-**constraint** is a pair

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

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 $c=\langle F: T'' \to T', \theta: \mathrm{Sign}(T') \to \Sigma \rangle$ iff $\theta(A)$ satisfies T' and is F-free; we write $A \models_\Sigma c$ in this case. []

A picture of the general situation in this definition may help:



In the Set-etc example, F: $T'' \to T'$ is the theory inclusion Etc: **Triv**$\to$**Set-of-Triv**, and θ: $\mathrm{Sign}(T') \to \Sigma$ is the signature morphism underlying the theory inclusion Etc: **Set-of-Triv**$\to$**Set-of-Triv-Etc**. For any Σ-algebra A, it makes sense to ask whether θA satisfies T'* and is F-free, as indeed Definition 30 does ask.

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 in [Thatcher, Wagner & Wright 79]. There are three main differences between our approach and the "initial restrictions" of [Reichel 80]: first, an initial restriction on an algebraic theory consists of a pair of subtheories, whereas we use a pair of theories with an arbitrary theory morphism between them, and a signature morphism from the target theory. The use of subtheories seems very natural, but we have been unable to prove that it gives rise to an institution, and conjecture that it does not do so; we also believe that the added generality may permit some interesting additional examples. The second difference is simply that we are doing our work over an arbitrary liberal institution. The third difference lies in the manner of adding constraints: whereas [Reichel 80] defines a "canon" to be an algebraic theory together with a set of 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. Historically the first work in this field seems to have been the largely unknown paper of [Kaphengst & Reichel 71], which apparently considered the case of a single chain of theory inclusions.

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

Definition 31: Let ϕ: $\Sigma \to \Sigma'$ 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 state the Satisfaction Condition for constraints.

Lemma 32: Constraint Satisfaction. If ϕ: $\Sigma \to \Sigma'$ is a signature morphism, if c is a Σ-constraint, and if B is a Σ'-model then

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

Given a liberal institution I, we can construct another institution having as its sentences both the sentences of I and also constraints.

Definition 33: Let I be an arbitrary liberal institution. Now construct the institution $C(I)$ whose theories contain both constraints and I-sentences as follows: the category of signatures of $C(I)$ is the category **Sign** of signatures of I; if Σ is an I-signature, then $\text{Sen}_{C(I)}(\Sigma)$ is the union of the set $\text{Sen}_I(\Sigma)$ of all Σ-sentences from I with the set of all Σ-constraints[9]; also $\mathbf{Mod}(\Sigma)$ is the same for $C(I)$ as for I; we use the concept of constraint translation in Definition 31 to define $\text{Sen}(\phi)$ on constraints; finally, satisfaction for $C(I)$ is as in I for Σ-sentences from I, and is as in Definition 30 for Σ-constraints. []

We now have the following.

Proposition 34: If I is a liberal institution, then $C(I)$ is an institution. []

This means that all the concepts and results of Section 3 can be applied to $C(I)$-theories that include constraints as well as sentences. Let us call such theories **constraining theories.** Thus, we get notions of presentation and of closure, as well as of theory. In particular, the Presentation and Closure Lemmas hold and we therefore get the following important result:

Theorem 35: Given a liberal institution with a [finitely] cocomplete category of signatures, its category of constraining theories is also [finitely] cocomplete.

Proof: Immediate from Theorems 27 and 34. []

Let us consider what this means for the equational institution. While the proof theory for inferring that an equation is in the closure of a set of equations is familiar, we have no such proof theory for constraints. However, this should be obtainable because a constraint corresponds to an induction principle plus some inequalities [Burstall & Goguen 81, Goguen & Meseguer 83]; in particular, the constraint that sets are to be interpreted freely will give us all the consequences of the induction principle for sets. In more detail, this constraint for sets demands that all elements of sort **set** be generated by the operators $\emptyset$, $\{\_\}$ and $\_ \cup \_$, and this can be expressed by a principle of structural induction over these generators (notice that this is not a first order axiom). The constraint also demands that two elements of sort **set** are *unequal* unless they can be proved equal using the given equations. We cannot express this distinctness with equations; one way to express it is by adding a new boolean-valued binary operator on the new sort, say $\equiv$, with some new equations such that $t \equiv t' = \text{false}$ iff $t \neq t'$, and also such that $\text{true} \neq \text{false}$ [Goguen 80].

Let now us 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.

---

[9] There are some foundational difficulties with the size of the closure of a constraint theory that we will ignore here; they can be solved by limiting the size of the category of signatures used in the original liberal institution.

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, inc(m)+n=inc(m+n), 1=inc(0).

Let Σ^N and Σ^{NP} be the signatures of **N** and **NP** respectively, and let F^N: **E**→**N** and F^{NP}: **N**→**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=false, 1=true, inc(false)=false, inc(true)=true, false+n=false, true+n=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 (F^N, F^{NP}) 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 $\langle f', \phi \rangle$, where f' is the inclusion of the empty theory **E** in the theory with 0, 1 and + with identity, associativity and commutativity equations, and ϕ is its inclusion in **NP**. This constraint is satisfied in all models that satisfy the constraint (F^N, F^{NP}). In this sense, the constraint on 0, 1 and + gives a derived induction principle. Further examples can be found in [Burstall & Goguen 81].

In general, the closure of a constraining presentation to a "constraining theory" adds new equations derivable by induction principles corresponding to the constraints; and it also adds some new constraints that correspond 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 parameterized theory whose meta-sort demands a commutative binary operator. The new constraints seem less essential. [Clark 78] and [McCarthy 80] discuss what may be a promising approach to the proof-theoretic aspect of constraining theories. Clark calls his scheme "predicate completion" and thinks of it as a method for infering new sentences under a "closed world" assumption (the common sense assumption that the information actually given about a predicate is all and only 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 is simply to infer the converse of any Horn clause. This is *sound* for the institutions of conditional equations and first order Horn clauses (in the sense that all the sentences thus obtained are true of the initial model); it is not clear that it is complete. McCarthy calls his scheme "circumscription" and is interested in its application in the context of full first order logic; he has shown that it is sound when there exist minimal models. It 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 $(F^N, \Sigma N^P)$. For example, 1+n=n contradicts the constraint; in fact, if we add it, we simply get an inconsistent constraining theory (it has no algebras).

4.3 Other Kinds of Constraint

Because a number of variations on the notion of data constraint have recently been proposed, it seems worthwhile to examine their relationship to the very general machinery developed in this paper. In fact, very little of Section 4.2 or Section 5 depends upon "F-freeness" in the definition of constraint satisfaction. This suggests weakening that notion. Recall that a Σ-algebra A satisfies a data constraint $c=\langle F: T \to T', \theta: \mathrm{Sign}(T') \to \Sigma \rangle$ iff θA satisfies T' and is F-free, which means that $\epsilon_{\theta A}$: $F^\$(F^*(\theta A)) \to A$ 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 corresponds to what has been called a "hierarchy constraint" [Broy, Dosch, Partsch, Pepper & Wirsing 79]; it means that no elements of θA are identified by the natural mapping from $F^{\$}(F^*(\theta A))$; this condition generalizes the "no confusion" condition of [Burstall & Goguen 82]. For $\epsilon_{\theta A}$ to be surjective corresponds to what has been called a "generating constraint" [Ehrig, Wagner & Thatcher 82]; it means that all elements of θA are generated by elements of $F^*(\theta A)$; this condition generalizes the "no junk" condition of [Burstall & Goguen 82]. Notice that these two together give that θA is F-free. Now here is the general notion:

Definition 36: Let I be an institution, let $\mathcal{M}$ be a class of model morphisms from I, let $c=(F: T \to T', \theta: \text{Sign}(T') \to \Sigma)$ be a constraint from I, and let A be a Σ-model from I. Then A **$\mathcal{M}$-satisfies** c iff θA satisfies T' and $\epsilon_{\theta A}$ lies in $\mathcal{M}$. []

In particular, for a liberal institution I, modifying the construction of $C(I)$ to use the notion of $\mathcal{M}$-satisfaction, gives an institution (generalizing Proposition 34), and theories in this institution will be as cocomplete as I is (generalizing Theorem 35). This means, for example, that we can glue together theories that use hierarchy constraints with the usual colimit constructions, and in particular, we can do the specification language constructions of Section 6.

5 Using More than One Institution

After the work of Section 4.2, we know how to express constraints in any liberal institution; in particular, we can use constraints in the equational institution to specify parameterized abstract data types. We can also give loose specifications in any institution. Liberality is a fairly serious restriction since 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. 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 how he can eat his cake and have it too. The basic idea is to precisely describe a relationship between two institutions, the second of which is liberal, in the form of an institution morphism, and then to permit constraints that use theories from the second institution as an additional kind of sentence in this "duplex" institution. There are also other uses for institution morphisms; in particular, we will use this concept in showing when a theorem prover for one institution is sound for theories from another institution.

5.1 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 (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 functions are the substance of an institution morphism $\mathcal{FOEQ} \to \mathcal{EQ}$; there are also some conditions that must be satisfied.

Definition 37: Let I and I' be institutions. Then an **institution morphism** Φ: $I \to I'$ consists of

1. a functor Φ: **Sign**$\to$**Sign'**,

2. a natural transformation α: Φ;Sen'$\Rightarrow$Sen, that is, a natural family of functions α_Σ: Sen'($\Phi(\Sigma)$)$\rightarrow$Sen(Σ), and
3. a natural transformation β: **Mod**$\Rightarrow\Phi$;**Mod'**, that is, a natural family of functors β_Σ: **Mod**(Σ)$\rightarrow$**Mod'**($\Phi(\Sigma)$),

such that the following **satisfaction condition** holds

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

for A a Σ-model from I and e' a $\Phi(\Sigma)$-sentence from I'. []

The reader may wish to verify that Φ: $\mathcal{FOEQ}\rightarrow\mathcal{EQ}$ as sketched in the first paragraph of this subsection really is an institution morphism.

Notice that an institution morphism Φ: $I\rightarrow I'$ induces a functor Φ: $\mathbf{Th}_I\rightarrow\mathbf{Th}_{I'}$ on the corresponding categories of theories by sending a Σ-theory T to the $\Phi(\Sigma)$-theory $\beta_\Sigma(T^*)^*$. We now have the following useful result, whose proof is omitted in this version of this paper:

Theorem 38: If Φ: $I\rightarrow I'$ is an institution morphism such that Φ: **Sign**$\rightarrow$**Sign'** is [finitely] cocontinuous, then Φ: $\mathbf{Th}_I\rightarrow\mathbf{Th}_{I'}$ is also [finitely] cocontinuous. []

Actually, a stronger result is true: Φ on theories preserves whatever colimits Φ preserves on signatures. By Theorem 38, if a large theory in the source institution is expressed as a colimit of smaller theories, the corresponding theory in the target institution can also be so expressed. Another reason for being interested in this result has to do with using theorem provers for one institution on theories in another (see below).

Definition 39: An institution morphism Φ: $I\rightarrow I'$ is **sound** iff for every signature Σ' and every Σ'-model A' from I', there are a signature Σ and a Σ-model A from I' such that $A'=\beta_\Sigma(A)$. []

This condition is clearly satisfied by the institution morphism from $\mathcal{FOEQ}$ to $\mathcal{EQ}$ discussed above.

Proposition 40: If Φ: $I\rightarrow I'$ is a sound institution morphism and if P is a set of Σ'-sentences from I', then a Σ'-sentence e' is in $P^\bullet$ iff αe is in $\alpha_\Sigma(P)^\bullet$, where Σ is a signature from I such that $\Sigma'=\Phi(\Sigma)$. []

We now know, for example, that the institution morphism from $\mathcal{FOEQ}$ to $\mathcal{EQ}$ permits using a theorem prover for first order logic with equality on equational theories.

5.2 Duplex Institutions

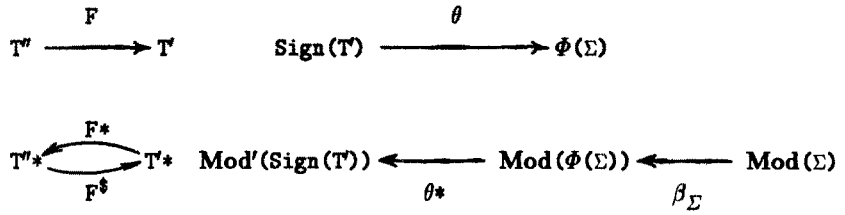
This subsection gives a construction for an institution whose theories can contain both sentences from a nonliberal institution I and constraints from a liberal institution I'. What is needed to make the construction work is an institution morphism Φ: $I\rightarrow I'$ expressing the relationship between the two institutions. This idea was introduced informally in [Goguen 82a] and [Burstall & Goguen 81]. Note that all the results of this section generalize to the notion of $\mathcal{M}$-satisfaction given in Definition 36, although they are stated for the case of data constraints, i.e., the case where $\mathcal{M}$ is the class of isomorphisms.

Definition 41: Let I' be a liberal institution, let Φ: $I\rightarrow I'$ be an institution morphism, and let Σ be a signature from I. Then a Σ-**duplex constraint** is a pair

$$c=(F: T''\rightarrow T', \theta: \mathrm{Sign}(T')\rightarrow\Phi(\Sigma)) ,$$

where F is a theory morphism from I' and θ is signature morphism from I. Furthermore, a Σ-model A from I **satisfies** the duplex constraint c iff $\theta(\beta_\Sigma(A))$ satisfies T' and is F-free; as usual, write $A \models_\Sigma c$. []

A 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 I.

Definition 42: Let Φ: $I \to I'$ be an institution with I' liberal, let Σ be a signature from I, let $c=\langle F, \theta\rangle$ be a Σ-duplex constraint, and let ϕ: $\Sigma \to \Sigma'$ be a signature morphism from I. Then the **translation** of c by ϕ is the Σ'-duplex constraint $\phi c=\langle F, \theta;\Phi(\phi)\rangle$. []

Lemma 43: Satisfaction. Let Φ: $I \to I'$ be an institution morphism with I' liberal, let Σ be a signature from I, let $c=\langle F, \theta\rangle$ be a Σ-duplex constraint, let ϕ: $\Sigma \to \Sigma'$ be a signature morphism from I, and let B be a Σ'-model from I. Then

$B \models_{\Sigma'} \phi c$ iff $\phi B \models_\Sigma c$. []

Definition 44: Let Φ: $I \to I'$ be an institution morphism with I' liberal and let Σ be a signature from I. Then the **duplex institution** over Φ, denoted $\mathcal{D}(\Phi)$ has: its signatures those from I; its Σ-sentences the Σ-sentences from I plus the Σ-duplex constraints; its Σ-models the Σ-models from I; and satisfaction is as defined separately for the sentences from I and the duplex constraints. []

Theorem 45: If I' is a liberal institution and Φ: $I \to I'$ is an institution morphism, then $\mathcal{D}(\Phi)$ is an institution. []

This result implies (by Theorem 27) that if the category of signatures of I is [finitely] cocomplete then so is the category of constraining theories that use as sentences both the sentences from I and the duplex constraints constructed using I', since this is the category $\mathbf{Th}_{\mathcal{D}(\Phi)}$ of theories of the duplex institution $\mathcal{D}(\Phi)$. Examples using this idea have been given in the specification languages Clear [Burstall & Goguen 81] and Ordinary [Goguen 82a, Goguen 82b].

There is another much simpler way to use an institution morphism Φ: $I \to I'$ to construct a new institution in which one simply permits sentences from either I or I'.

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

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

For example, it may be convenient to use already existing equational theories when construcing new first order theories.

5.3 Multiplex Institutions

Actually, we can use many different institutions all at once, some of them liberal institutions for various kinds of constraints, and some of them not necessarily liberal, for expressive power; all that is needed is a morphism to each from the basic institution. Let I be an institution and let Φ_i: $I \to I_i$ be institution morphisms for i=1,...,m+n, where I_i are liberal for i=1,...,n. Then we define $I(\Phi_1,...,\Phi_n; \Phi_{n+1},...,\Phi_{n+m})$ to be the institution with: signatures those from I; sentences either sentences from I, or else constraints $(F: T'' \to T', \theta: \text{Sign}(T') \to \Phi_i(\Sigma))$ for θ, T'', T' from I_i for some $1 \leq i \leq n$, or else pairs $(T, \theta: \text{Sign}(T) \to \Phi_i(\Sigma))$ with θ, T from I_i for some $n+1 \leq i \leq n+m$; with its models those of I; and with satisfaction as usual for the I-sentences and for the constraints, and as in Definition 46 for the others. Notice that the liberal insititutions can each use a different collection $\mathcal{M}$ of morphisms to define constraint satisfaction. One can even use the same liberal institution with different classes $\mathcal{M}$. In particular, this means that one can glue together (with colimits) 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.

6 Specification

There is now enough technical machinery so that we can consider some operations on theories that are useful in constructing specifications. For more detailed explanations and examples, see the Clear language and its semantics [Burstall & Goguen 80, Burstall & Goguen 81]; similar ideas appear in several other specification languages.

A specification can be given directly by a presentation in the appropriate institution, taken to denote the theory which is the closure of the presentation. More complex theories can be built up from smaller ones by applying combination operations. A specification language, such as Clear, gives a notation for these operators on theories. An alternative approach is to consider operations on categories of models instead of operations on theories. This has been pursued by [Thatcher, Wagner & Wright 79] and should also fit into the institutional framework. Another approach favoured by some is to consider operations on the presentations; however, we prefer a more "semantic" approach.

6.1 Combining Theories

Suppose that we have separately specified a theory **Bool** of truth values and a theory **Nat** of natural numbers, each with appropriate operators. We would have to combine these to get, say, **Bool + Nat**, before defining an extra operator such as $\leq$: **nat x nat→bool**, whose definition involves sorts and operators taken from each theory. (We call adding such extra operators or sorts **enriching** a theory).

There are some issues about the most convenient definition of the combine operation. Suppose that we combine theories T and T' in the equational institution. Should we take the union of their sorts, operators and equations? This would handle correctly cases where they both use sorts like **bool** which should be the same in each (such cases are quite common). Or should we take the disjoint union? This would correctly handle cases where there is an operator appearing in each with the same name, but inadvertently so; for example, the writers of the two

specifications may have used the same name by chance (this is not unlikely with large specifications).

Clear allowed shared subtheories, while still attempting to avoid the problems associated with a chance reuse of the same name. Each theory keeps track of the theories from which it is constructed, so that, for example, it is known that T + T' and T' + T'' both contain T. If these two theories are themselves combined, then the sorts and operators in T will not be duplicated. But if the same name is used by chance in T' and in T'', then separate copies will be kept, disjoint union style. The mechanism is to use not simple theories but rather "based theories," which are cones in the category of theories. The tip of the cone is the current theory, and the diagram forming the base of the cone shows which theories were used to build it up (see [Burstall & Goguen 80] for details). The coproduct of two such cones gives the required combine operation, written T + T'.

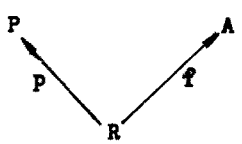
This works, but it is rather sophisticated. Another approach is to ensure that each sort or operator gets a unique name when it is declared; then we can take a simple union of the theories. But union is a set theoretic notion, while the institutional framework uses an arbitrary category of signatures. The solution seems to be to use a subcategory of signature **inclusions**, that is, a full subcategory that is a poset and whose morphisms are called inclusions, and should all be monics. We then no longer need to use cones, and can simply take as T + T' their coproduct in the subcategory of inclusions. Enrichments will give rise to inclusions. (This proposed simpler semantics needs further study.)

6.2 Parameterised Specifications

We often want to write a specification using some "unknown" sorts and operators, that can later be instantiated (as with generic packages in Ada). Then we can develop a library of such parameterised specifications. Thinking of them as specification-building operations, or as specification-valued procedures, we can design a specification language as a functional language whose data elements are theories. An attractive mathematical framework for this uses the pushout construction to apply a parameterised specification to an argument, as we shall explain.

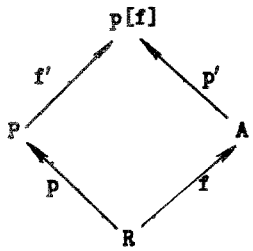
A parameterised specification may be taken to denote a theory, P, having a distinguished subtheory, R; that is, there is an inclusion morphism p: R→P. The subtheory R shows which part of P is the parameter part to be instantiated. For example, P might be a theory about ordered sequences of elements, and R might be the subtheory of the elements and their ordering. By choosing different instances of R, for example natural numbers with their usual $\leq$ ordering, or sets with the inclusion ordering, we get different instances of P, namely ordered sequences of numbers, or ordered sequences of sets.

What do we mean by saying that a theory A is an instance of a theory R? For equational theories we need to map the sorts and operators of R to sorts and operators of A, in such a way that equations of R are preserved. In general we need a theory morphism f: R→A. We may call R the **requirement**, f the **fitting morphism**, and A the **actual parameter**. Thus we have the diagram



The fitting morphism serves to *bind* the material in the actual parameter to the "required" material in the requirement theory. Since the fitting morphism must be a theory morphism, just its existence implies that the actual parameter satisfies the axioms given in the requirement theory, once its syntax has been changed in accord with the bindings given by the fitting morphism.

What do we mean by applying the parameterised theory P to the argument theory A (or more accurately, by applying p to f)? We want a theory that includes what is in P, as well as the new material from A, suitably instantiated by identifying the R part of P with the corresponding elements in A (this correspondence being given by f). It turns out that the pushout of the above diagram is the required categorical construction; we can think of it as yielding the sum of P and A, amalgamating their R parts. (For the equational institution, it has been *proven* that theory really do behave this way [Goguen & Burstall 78].) Writing p[f] for the result of the application, we have the pushout diagram



We would expect p and p' to both be inclusions in this diagram.

6.3 Enrichments

We can enrich a specification to get a larger specification. In the case of equational theories, this is done by adding new sorts, operators and equations; for example, adding to **Bool + Nat** the new operator $\leq$: **nat x nat→bool** with its defining equations, such as $0 \leq n = \text{true}$. Note that the new material does not itself constitute a theory, so we cannot use combine here.

However, enrichment is really just a special case of applying a parameterised specification. Suppose that we wish to enrich a theory T. The new operator declarations may refer to sorts in T, and the new equations to operators in T. So there is a minimal subtheory of T needed to support this new material, say T', with $T' \subseteq T$, that is, with an inclusion i: T'→T. T' together with the new material forms a theory T'', with $T' \subseteq T''$. We may think of the enrichment as an abbreviation for an inclusion j: T'→T''. The result of the enrichment is just the pushout of i and j, or considering j as a parameterised specification and i as the fitting morphism, it is j[i].

6.4 An Analogy

This section has applied ideas of the preceding sections to problems of program specification, including theory combination, theory environments, and parameterized theories. The discussion follows an analogy [Burstall & Goguen 77] between specification languages and programming languages, in which theories are treated as values, parameterized theories correspond to procedures, based theories are environments, and requirement theories correspond to types. This last part of the analogy can be seen as an application of the "correspondence principle" of [Landin 66].

Just as declarations for variables and procedures create environments containing local values for identifiers in a programming language, so do declarations in a specification language create environments having local values. Whereas an environment in a programming language is just an assignment of values (and possibly types) to identifiers, environments in a specification language must also keep track of relationships between theories, so that there is not only an assignment of theories to identifiers, but also a collection of theory inclusions among the given theories, indicating that some theories are contained in others. The basic semantic insight is that to put theories together, whether for combining, enriching, or for applying a parameterized theory to an actual theory, one can simply take colimits of appropriate diagrams.

7 Summary

We have formalized the intuitive notion of a "logical system" or "abstract model theory" with the abstract notion of an institution, and have shown that institutions whose signatures have finite colimits are suitable for use in programming methodology. In a liberal institution, the forgetful functors induced by theory morphisms, have corresponding free functors, and data types can be defined by "constraints," which are abstract induction principles. Allowable kinds of constraint include data, generating, and hierarchy. We have introduced the notion of an institution morphism and shown how to use it in determining 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 formalize the simultaneous use of more than one institution; this permits combining the greater expressive power of nonliberal institutions with the data type definition capability of liberal institutions. Finally, we showed how to formulate a number of basic specification language constructions in an arbitrary institution having finite colimits.

References

[Arbib & Manes 75]
Arbib, M. A. and Manes, E.
Arrows, Structures and Functors.
Academic Press, 1975.

[Aubin 76] Aubin, R.
Mechanizing Structural Induction.
PhD thesis, University of Edinburgh, 1976.

[Barwise 74] Barwise, J.
Axioms for Abstract Model Theory.
Annals of Mathematical Logic 7:221-265, 1974.

[Benabou 68] Benabou, J.
Structures Algebriques dans les Categories.
Cahiers de Topologie et Geometrie Differentiel 10:1-126, 1968.

[Birkhoff & Lipson 70]
Birkhoff, G. and Lipson, J.
Heterogeneous Algebras.
Journal of Combinatorial Theory 8:115-133, 1970.

[Bloom 76] Bloom, S. L.
Varieties of Ordered Algebras.
Journal of Computer and System Sciences 13:200-212, 1976.

[Boyer & Moore 80]
Boyer, R. and Moore, J. S.
A Computational Logic.
Academic Press, 1980.

[Broy, Dosch, Partsch, Pepper & Wirsing 79]
Broy, M., Dosch, N., Partsch, H., Pepper, P. and Wirsing, M.
Existential Quantifiers in Abstract Data Types.
In *Proceedings, 6th ICALP*, pages 73-87. Springer-Verlag, 1979.
Lecture Notes in Computer Science, volume 71.

[Burstall & Goguen 77]
Burstall, R. M. and Goguen, J. A.
Putting Theories together to Make Specifications.
Proceedings, Fifth International Joint Conference on Artificial Intelligence 5:1045-1058, 1977.

[Burstall & Goguen 78]
Burstall, R. M. and Goguen, J. A.
Semantics of Clear.
1978.
Unpublished notes handed out at the Symposium on Algebra and Applications, Stefan Banach Center, Warszawa, Poland.

[Burstall & Goguen 80]
Burstall, R. M., and Goguen, J. A.
The Semantics of Clear, a Specification Language.
In *Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification*, pages 292-332. Springer-Verlag, 1980.
Lecture Notes in Computer Science, Volume 86.

[Burstall & Goguen 81]
Burstall, R. M. and Goguen, J. A.
An Informal Introduction to Specifications using Clear.
In Boyer, R. and Moore, J (editor), *The Correctness Problem in Computer Science*, pages 185-213. Academic Press, 1981.

[Burstall & Goguen 82]
Burstall, R. M. and Goguen, J. A.
Algebras, Theories and Freeness: An Introduction for Computer Scientists.
In *Proceedings, 1981 Marktoberdorf NATO Summer School*, . Reidel, 1982.

[Burstall & Rydeheard 80]
Burstall, R. M. and Rydeheard, D. E.
Signatures, Presentations and Theories: a Monad Approach.
Technical Report, Computer Science Department, University of Edinburgh, 1980.

[Clark 78] Clark, K. L.
Negation as Failure.
In H. Gallaire and J. Minker (editor), *Logic in Data Bases*, pages 293-322. Plenum Press, 1978.

[Cohn 65] Cohn, P. M.
Universal Algebra.
Harper and Row, 1965.
Revised edition 1980.

[Ehrich 78] Ehrich, H.-D.
On the Theory of Specification, Implementation and Parameterization of Abstract Data Types.
Technical Report, Forschungsbericht, Dortmund, 1978.

[Ehrig, Kreowski, Rosen & Winkowski 78]
Ehrig, E., Kreowski, H.-J., Rosen, B. K. and Winkowski, J.
Deriving Structures from Structures.
In *Proceedings, Mathematical Foundations of Computer Science*, . Springer-Verlag, Zakopane, Poland, 1978.
Also appeared as technical report RC7046 from IBM Watson Research Center, Computer Sciences Dept.

[Ehrig, Wagner & Thatcher 82]
Ehrig, H., Wagner, E. and Thatcher, J.
Algebraic Specifications with Generating Constraints.
Technical Report, IBM Research Center, Yorktown Heights, New York, 1982.
Draft report.

[Gabriel & Ulmer 71]
Gabriel, P. and Ulmer, F.
Lokal Prasentierbare Kategorien.
Springer-Verlag, 1971.
Springer Lecture Notes in Mathematics, vol. 221.

[Gerhard, Musser et al. 79]
Gerhard, S. L., Musser, D. R., Thompson, D. H., Baker, D. A., Bates, R. W., Erickson, R. W., London, R. L., Taylor, D. G., and Wile, D. S.
An Overview of AFFIRM: A Specification and Verification System.
Technical Report, USC Information Sciences Institute, Marina del Rey, CA, 1979.

[Goguen 71] Goguen, J.
Mathematical Foundations of Hierarchically Organized Systems.
In E. Attinger (editor), *Global Systems Dynamics*, pages 112-128. S. Karger, 1971.

[Goguen 74] Goguen, J. A.
Semantics of Computation.
In *Proceedings, First International Symposium on Category Theory Applied to Computation and Control*, pages 234-249. University of Massachusetts at Amherst, 1974.
Also published in Lecture Notes in Computer Science, Vol. 25., Springer-Verlag, 1975, pp. 151-163.

[Goguen 77] Goguen, J. A.
Abstract Errors for Abstract Data Types.
In *IFIP Working Conference on Formal Description of Programming Concepts*, . MIT, 1977.
Also published by North-Holland, 1979, edited by P. Neuhold.

[Goguen 78] Goguen, J. A.
Order Sorted Algebra.
Technical Report, UCLA Computer Science Department, 1978.
Semantics and Theory of Computation Report No. 14; to appear in *Journal of Computer and System Science.*

[Goguen 80] Goguen, J. A.
How to Prove Algebraic Inductive Hypotheses without Induction: with applications to the correctness of data type representations.
In W. Bibel and R. Kowalski (editor), *Proceedings, 5th Conference on Automated Deduction*, pages 356-373. Springer-Verlag, Lecture Notes in Computer Science, Volume 87, 1980.

[Goguen 82a] Goguen, J. A.
Ordinary Specification of Some Construction in Plane Geometry.
In Staunstrup, J. (editor), *Proceedings, Workshop on Program Specification*, pages 31-46. Springer-Verlag, 1982.
Lecture Notes in Computer Science, Volume 134.

[Goguen 82b] Goguen, J. A.
Ordinary Specification of KWIC Index Generation.
In Staunstrup, J. (editor), *Proceedings, Aarhus Workshop on Specification*, pages 114-117. Springer-Verlag, 1982.
Lecture Notes in Computer Science, Volume 134.

[Goguen & Burstall 78]
Goguen, J. A. and Burstall, R. M.
Some Fundamental Properties of Algebraic Theories: a Tool for Semantics of Computation.
Technical Report, Dept. of Artificial Intelligence, University of Edinburgh, 1978.
DAI Research Report No. 5; to appear in *Theoretical Computer Science.*

[Goguen & Ginali 78]
Goguen, J. A. and Ginali, S.
A Categorical Approach to General Systems Theory.
In Klir, G. (editor), *Applied General Systems Research*, pages 257-270. Plenum, 1978.

[Goguen & Meseguer 81]
Goguen, J. A. and Meseguer, J.
Completeness of Many-sorted Equational Logic.
SIGPLAN Notices 16(7):24-32, July, 1981.
Also appeared in *SIGPLAN Notices*, January 1982, vol. 17, no. 1, pages 9-17; extended version as SRI Technical Report, 1982, and to be published in *Houston Journal of Mathematics.*

[Goguen & Meseguer 83]
Goguen, J. A. and Meseguer, J.
An Initiality Primer.
In Nivat, M. and Reynolds, J. (editor), *Application of Algebra to Language Definition and Compilation*, . North-Holland, 1983.
To appear.

[Goguen & Parsaye-Ghomi 81]
Goguen, J. A. and Parsaye-Ghomi, K.
Algebraic Denotational Semantics using Parameterized Abstract Modules.
In J. Diaz and I. Ramos (editor), *Formalizing Programming Concepts*, pages 292-309. Springer-Verlag, Peniscola, Spain, 1981.
Lecture Notes in Computer Science, volume 107.

[Goguen & Tardo 79]
Goguen, J. A. and Tardo, J.
An Introduction to OBJ: A Language for Writing and Testing Software Specifications.
In *Specification of Reliable Software*, pages 170-189. IEEE, 1979.

[Goguen, Thatcher & Wagner 78]
Goguen, J. A., Thatcher, J. W. and Wagner, E.
An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types.
In R. Yeh (editor), *Current Trends in Programming Methodology*, pages 80-149. Prentice-Hall, 1978.
Original version, IBM T. J. Watson Research Center Technical Report RC 6487, October 1976.

[Goguen, Thatcher, Wagner & Wright 75a]
Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B.
An Introduction to Categories, Algebraic Theories and Algebras.
Technical Report, IBM T. J. Watson Research Center, Yorktown Heights, N. Y., 1975.
Research Report RC 5369.

[Goguen, Thatcher, Wagner & Wright 75b]
Goguen, J. A., Thatcher, J. W., Wagner, E. and Wright, J. B.
Abstract Data Types as Initial Algebras and the Correctness of Data Representations.
In *Computer Graphics, Pattern Recognition and Data Structure*, pages 89-93. IEEE, 1975.

[Goldblatt 79] Goldblatt, R.
Topoi, The Categorial Analysis of Logic.
North-Holland, 1979.

[Higgins 63] Higgins, P. J.
Algebras with a Scheme of Operators.
Mathematische Nachrichten 27:115-132, 1963.

[Hoare 72] Hoare, C. A. R.
Proof of Correctness of Data Representation.
Acta Informatica 1:271-281, 1972.

[Kaphengst & Reichel 71]
Kaphengst, H. and Reichel, H.
Algebraische Algorithemtheorie.
Technical Report WIB Nr. 1, VEB Robotron, Zentrum fur Forschung und Technik, Dresden, 1971.
In German.

[Landin 66] Landin, P. J.
The Next 700 Programming Languages.
Communications of the Association for Computing Machinery 9, 1966.

[Lawvere 63] Lawvere, F. W.
Functorial Semantics of Algebraic Theories.
Proceedings, National Academy of Sciences 50, 1963.
Summary of Ph.D. Thesis, Columbia University.

[Lawvere 64] Lawvere, F. W.
An Elementary Theory of the Category of Sets.
Proceedings, National Academy of Sciences, U.S.A. 52:1506-1511, 1964.

[Levitt, Robinson & Silverberg 79]
Levitt, K., Robinson, L. and Silverberg, B.
The HDM Handbook.
Technical Report, SRI, International, Computer Science Lab, 1979.
Volumes I, II, III.

[MacLane 71] MacLane, S.
Categories for the Working Mathematician.
Springer-Verlag, 1971.

[MacLane & Birkhoff 67]
MacLane, S. and Birkhoff, G.
Algebra.
Macmillan, 1967.

[Mahr & Makowsky 82a]
Mahr, B. and Makowsky, J. A.
An Axiomatic Approach to Semantics of Specification Languages.
Technical Report, Technion, Israel Institute of Technology, 1982.
Extended Abstract.

[Mahr & Makowsky 82b]
Mahr, B. and Makowsky, J. A.
Characterizing Specification Languages which Admit Initial Semantics.
Technical Report, Technion, Israel Institute of Technology, February, 1982.
Technical Report #232.

[McCarthy 80] McCarthy, J.
Circumscription - A Form of Non-Monotonic Reasoning.
Artificial Intelligence 13(1,2):27-39, 1980.

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

[Reichel 80] Reichel, H.
Initially Restricting Algebraic Theories.
Springer Lecture Notes in Computer Science 88:504-514, 1980.
Mathematical Foundations of Computer Science.

[Shostak, Schwartz & Melliar-Smith 81]
Shostak, R., Schwartz, R. & Melliar-Smith, M.
STP: A Mechanized Logic for Specification and Verification.
Technical Report, Computer Science Lab, SRI International, 1981.

[Tarski 44] Tarski, A.
The Semantic Conception of Truth.
Philos. Phenomenological Research 4:13-47, 1944.

[Thatcher, Wagner & Wright 79]
Thatcher, J. W., Wagner, E. G. and Wright, J. B.
Data Type Specification: Paramerization and the Power of Specification Techniques.
In *Proceedings of 1979 POPL*, . ACM, 1979.

[Wright, Thatcher, Wagner & Goguen 76]
Wright, J. B., Thatcher, J. W., Wagner, E. G. and Goguen, J. A.
Rational Algebraic Theories and Fixed-Point Solutions.
Proceedings, 17th Foundations of Computing Symposium :147-158, 1976.
IEEE.

Table of Contents