LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

A Study in the Foundations of Programming Methodology: Specifications, Institutions, Charters and Parchments

by

Joseph A.Goguen and R.M.Burstall

LFCS Report Series ECS-LFCS-86-10
August 1986

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

To appear in Proc. of Summer Workshop on Category Theory & Computer Programming, University of Surrey, 1985.

A Study in the Foundations of Programming Methodology: Specifications, Institutions, Charters and Parchments

by
Joseph A. Goguen and R. M. Burstall
SRI International and the University of Edinburgh
and
Center for the the Study of Language and Information
Stanford University

Abstract

The theory of institutions formalizes the intuitive notion of a "logical system." Institutions were introduced (1) to support as much computer science as possible *independently* of the underlying logical system, (2) to facilitate the transfer of results (and artifacts such as theorem provers) from one logical system to another, and (3) to permit combining a number of different logical systems. In particular, programming-in-the-large (in the style of the Clear specification language) is available for any specification or "logical" programming language based upon a suitable institution. Available features include generic modules, module hierarchies, "data constraints" (for data abstraction), and "multiplex" institutions (for combining multiple logical systems). The basic components of an **institution** are: a category of **signatures** (which generally provide symbols for constructing sentences); a set (or category) of **Σ-sentences** for each signature Σ; a category (or set) of **Σ-models** for each Σ; and a **Σ-satisfaction** relation, between Σ-sentences and Σ-models, for each Σ. The intuition of the basic axiom for institutions is that *truth (i.e., satisfaction) is invariant under change of notation*. This paper enriches institutions with sentence morphisms to model proofs, and uses this to explicate the notion of a logical programming language.

To ease constructing institutions, and to clarify various notions, this paper introduces two further concepts. A **charter** consists of an adjunction, a "base" functor, and a "ground" object; we show that "chartering" is a convenient way to "found" institutions. **Parchments** provide a notion of sentential syntax, and a simple way to "write" charters and thus get institutions. Parchments capture the insight that *the syntax of logic is an initial algebra*. Everything is illustrated with the many-sorted equational institution. Parchments also explicate the sense of finitude that is appropriate for specifications. Finally, we introduce **generalized institutions**, which generalize both institutions and Mayoh's "galleries", and we introduce corresponding generalized charters and parchments.

1 Introduction

The major theme of this paper is the study of abstract concepts of "logical system" that are suitable as foundations for programming methodology. The basic concept is that of an "institution", which provides suitably interrelated notions of sentence, model and satisfaction; several different but equivalent formulations of this concept are given, thus providing evidence for its naturality[1]. This approach avoids commitment to particular logical systems by doing constructions once and for all, over any suitable logical system; in particular, various modularization techniques introduced in the specification language Clear [Burstall & Goguen 77, Burstall & Goguen 80, Burstall & Goguen 81] are possible in the general setting, and apply not only to specification languages, but also to "logical" programming languages that are based upon pure logical systems. The concepts of "charter" and "parchment" provide ways to create institutions, and they also capture the intuition that sentences are constructed from more basic syntactic elements[2]. "Generalized" notions of institution, charter, and parchment enlarge applicability to systems such as databases.

This paper presupposes some familiarity with category theory (for which see [Mac Lane 71]) and often refers to [Goguen & Burstall 85] in preference to repeating details which are given there about institutions. By way of basic notation, categories are underlined, $|\underline{C}|$ denotes the class of objects of $\underline{C}$, f;g denotes the composition of morphisms f and g in diagramatic order, 1_A denotes the identity at an object A, and $\underline{C}^{op}$ denotes the opposite category of $\underline{C}$.

1.1 What is a Specification?

A specification is a *finite* text that should be readable at least by humans, and preferably by computers. Thus, some unsuitable notions of specification include:

- a set (even finite) of infinitary sentences;
- a theory (i.e., a class of sentences closed under semantic entailment);
- a class of models (whether or not closed); and
- an equivalence relation on the class of all models.

It would not be helpful for a program designer to give such a thing to a programmer. Athough all of these have been suggested in the literature, they all fail to be finitely

---

[1] This is similar to arguments for the Church-Turing thesis.

[2] For non-speakers of English, we provide the following glossary of the conventional uses of our technical terms: an institution is an established organization, such as a bank or a scientific society; a charter is a legal document creating such an institution; and a parchment is an ancient form of paper used for important documents.

readable, and seem to be examples of over-abstraction[3]. We will suggest an explication of "specification" later in this paper, using the parchment concept to formalize finitude.

To clarify the concepts itemized above, a specification is a finite text which determines a theory, which determines a class of models, where the **theory** is a (usually) infinite set of sentences. The model class contains all models of the specification, and the theory contains all sentences that are true of all models of the specification.

We claim that *putting together small specifications to describe complex models* is the essence of a specification language; the rest has to do with the particular brands of syntactic sugar and underlying logic that are used. The motivation is of course to make it easier to write specifications for large and complex programs. The following are some tricks that were introduced in the specification language Clear [Burstall & Goguen 77, Burstall & Goguen 80, Burstall & Goguen 81] for these purposes:

- use *colimits* to put theories together
- use *diagrams* as environments to keep track of shared subtheories
- use *data constraints*[4] to define particular structures (i.e., abstract data types)
- use *pushouts* to apply generic theories to their "actual" arguments, and
- use *theory morphisms* to describe the bindings of actuals at interface theories (also called "requirement" theories).

These ideas have been implemented in the programming/specification language OBJ2 [Futatsugi, Goguen, Jouannaud & Meseguer 85]; in this sense, OBJ2 is an implementation of Clear. More generally, these ideas give programming-in-the-large for any programming language that is purely based upon some logical system[5], including:

- OBJ2, with equational logic
- Eqlog [Goguen & Meseguer 86a], with Horn clause logic with equality
- pure Prolog [van Emden & Kowalski 76, Lloyd 84], with Horn clause logic; and
- FOOPS [Goguen & Meseguer 86b], with reflective equational logic[6].

We will later suggest a general notion of "logical" programming language to capture these

---

[3]However, it does seem reasonable to give a finite text which describes the construction of a single model in set theory, as in VDM [Bjorner & Jones 78] or Z [Abrial, Schuman & Meyer 79]; it seems an interesting problem to relate these approaches to institutions.

[4]See also the canons of [Reichel 84].

[5]They can even be applied to conventional languages, such as Ada; see [Goguen 86].

[6]FOOPS is a Functional Object Oriented Programming System.

and other examples. Such languages can have abstract data types, generic modules, and integration of specifications with executable code (i.e, "wide spectrum" capability). This makes all of the following easier:

- reading and understanding code
- debugging code
- proving code
- implementing the language, and
- providing a rigorous mathematical theory of the language.

1.2 Acknowledgements

We wish to thank the institutions at which we have worked, SRI International, the University of Edinburgh, and the Center for the Study of Language and Information at Stanford University, plus the institutions that have sponsored the work: in the U.S., the National Science Foundation, the Office of Naval Research (Contracts N00014-82-C-0333 and N00014-85-C-0417), and the System Development Foundation for a gift supporting the work at CSLI; and in the U.K., the Science and Engineering Research Council and British Petroleum; also thanks to first order logic and equational logic, where we started. Very special thanks to Andrzej Tarlecki for many helpful comments and ideas, and to José Meseguer for a careful reading of the manuscript and several valuable suggestions for its improvement; thanks also to Brian Mayoh and Christoph Beierle for their valuable suggestions.

2 Institutions

The original motivation for developing institutions was to do the "Clear tricks" once and for all, over any (suitable) logical system; see [Burstall & Goguen 80], where institutions were called "languages". This would make these tricks available for a variety of specification and logical programming languages. More recently, we have been exploring the use of institutions to provide general foundations for other areas of computer science. Intuitively, an **institution** is a formalization of the notion of "logical system" having the following:

- **signatures**, which generally provide vocabularies for sentences
- **Σ-sentences**, for each signature Σ
- **Σ-models**, for each signature Σ
- a **Σ-satisfaction** relation, of Σ-sentences by Σ-models, for each signature Σ, and
- **signature morphisms**, which describe changes of notation, with corresponding transformations for sentences and models.

In addition, one may well want homomorphisms of models and/or morphisms of sentences (which may be seen as "proofs"). One view is that institutions generalize classical model theory by *relativizing* it over signatures. This intuition is stated in the following slogan:

Truth is invariant under change of notation.

This subject is closely related to "abstract model theory" as studied by logicians, e.g., [Barwise 74].

Now the formalization:

Definition 1: An **institution** I consists of:

- a category Sign of **signatures**
- a functor Mod: Sign$\to$Cat^{op} giving **Σ-models** and **Σ-morphisms**
- a functor Sen: Sign$\to$Cat giving **Σ-sentences** and **Σ-proofs**
- a **satisfaction** relation $\models_\Sigma \subseteq |\mathrm{Mod}(\Sigma)| \times |\mathrm{Sen}(\Sigma)|$ for each $\Sigma \in |\mathrm{Sign}|$

such that

- **satisfaction:** $m' \models_{\Sigma'} \mathrm{Sen}(\phi)s$ iff $\mathrm{Mod}(\phi)m' \models_\Sigma s$ for each $m' \in |\mathrm{Mod}(\Sigma')|$, $s \in |\mathrm{Sen}(\Sigma)|$, $\phi\colon \Sigma \to \Sigma'$ in Sign, and
- **soundness:** $m \models_\Sigma s$ and $s \to s' \in \mathrm{Sen}(\Sigma)$ imply $m \models_\Sigma s'$ for $m \in |\mathrm{Mod}(\Sigma)|$.

□

Actually, most of what we do uses a simpler definition of institution, with sentence functor Sen: Sign$\to$Set, and thus without proofs and without need for the soundness axiom (Section 5.1 is a major exception). One might also want to simplify models to eliminate model-morphisms, thus using a model functor Mod: Sign$\to$Set^{op}. Thus, there are altogether four minor variants of the institution concept. We shall use the word "simplest" for the variant where both functors are Set-valued, and "simple" for the common variant with model morphisms but without sentence morphisms. We note in passing that there is also a definition as a functor into a comma category of "twisted relations",

I: Sign$\to$(U$\downarrow$/U$\uparrow$) ,

where U: Cat$\to$Set is the forgetful functor; see [Goguen & Burstall 85] for details.

The following are examples of institutions:

- first order logic
- first order logic with equality
- Horn clause logic
- Horn clause logic with equality
- equational logic

- order-sorted equational logic
- continuous equational logic

each in both one and many-sorted versions[7]. A lot of interesting computer science can be done independently of the choice of institution; for example, an institutional study of the notion of implementation is given by [Beierle & Voss 85], free constructions (which are "closed worlds" in the jargon of Artificial Intelligence) are studied by [Tarlecki 84], and observational equivalence of software modules is studied by [Sanella & Tarlecki 85a, Sanella & Tarlecki 85b].

This paper may seem to present an overabundance of variations on the institution theme; but, after all, that is its purpose! Five equivalent formulations are actually mentioned:

1. the basic formulation of Definition 1
2. the twisted relation definition mentioned above
3. the extranatural transformation definition in Section 2.2 below
4. a "room" definition in Section 5, and
5. a diagram and comma category definition also in Section 5,

and each of these has four minor variants. The last two actually present institutions as a special case of "generalized" institutions. In addition, we present a two step process for founding institutions, involving charters and parchments; there are also generalized charters and parchments. The major unstated theorem of this paper is that *all definitions of institution are equivalent* (modulo the four minor variants).

2.1 The Equational Institution

This subsection outlines the equational institution. We treat the many-sorted case (instead of the one-sorted case) because it presents some interesting features, in particular, the need to explicitly declare variables for equations; [Goguen & Meseguer 85] show that the usual rules of equational deduction for the one-sorted case are unsound if used for many-sorted deduction, and that this can be fixed by adding variable declarations to equations. Also, we use the many-sorted algebra notation of [Goguen 74], which systematically employs sort-indexed sets. A direct proof of the satisfaction condition for many-sorted equational logic is given in [Goguen & Burstall 85]. While it is certainly not deep, it is a bit of effort; moreover, this effort is unnecessary, since satisfaction follows automatically from the chartering construction given in Section 3. Now here are the constituents:

---

[7] Most of the proofs that these are institutions can be found in [Goguen & Burstall 85]; however, these are for the simple notion of institution without sentence morphisms.

- Sign is the category SigAlg of signatures for many-sorted algebra, defined as follows:
  - its objects, the **signatures**, are pairs $\langle S,\Sigma\rangle$ where $\Sigma=\{\Sigma_{w,s} \mid w\in S^*, s\in S\}$ where each $\Sigma_{w,s}$ is a set, and
  - **signature morphisms** $\langle S,\Sigma\rangle\rightarrow\langle S',\Sigma'\rangle$ are pairs $\langle\phi,\psi\rangle$, where ϕ: $S\rightarrow S'$ and ψ: $\Sigma\rightarrow\Sigma'$ where $\psi = \langle\psi_{w,s}: \Sigma_{w,s}\rightarrow\Sigma'_{\phi(w),\phi(s)} \mid w\in S^*, s\in S\rangle$.

  For some purposes, it is useful to assume that signatures consist of *disjoint* sets of symbols.
- Mod is the functor Alg sending a signature Σ to the category $\underline{Alg}(\Sigma)$ of Σ-algebras and Σ-homomorphisms. If Σ has sort set S, then a Σ-algebra A consists of an S-sorted set $\langle A_s \mid s\in S\rangle$ of carrier sets and a function A_σ: $A_w\rightarrow A_s$ for each $\sigma\in\Sigma_{w,s}$ where $A_w=A_{s1}\times\ldots A_{sn}$ when $w=s1\ldots sn$ and $A_\lambda=1$, some one pointed set[8]. A Σ-homomorphism h: $A\rightarrow B$ is an S-sorted set $\langle h_s: A_s\rightarrow B_s \mid s\in S\rangle$ preserving each operation in Σ. Then $Alg(\sigma)$: $\underline{Alg}(\Sigma')\rightarrow\underline{Alg}(\Sigma)$ for σ: $\Sigma\rightarrow\Sigma'$ is a functor; we may write A'^{σ} for $Alg(\sigma)(A')$.
- $Sen(\Sigma)$ is the set of all Σ-equations, where a Σ-equation is a triple $\langle\mathcal{V},t1,t2\rangle$ where $\mathcal{V}$ is a collection $\langle\mathcal{V}_s \mid s\in S\rangle$ of finite sets of variable symbols, and t1,t2 are Σ-terms of the same sort with variables from $\mathcal{V}$.
- Satisfaction is the usual satisfaction of an equation by an algebra.

2.2 Institutions as Extranatural Transformations

There is also (thanks to a suggestion from Gavin Wraith) an elegant formulation of institutions as extranatural transformations. Let S: $\underline{C}^{op}\times\underline{C}\rightarrow\underline{B}$ be a functor and let b be an object of $\underline{B}$. Then an **extranatural transformation**[9] (also called a **wedge** or a **supernatural transformation**), denoted

$$\alpha: S \dashrightarrow b\ ,$$

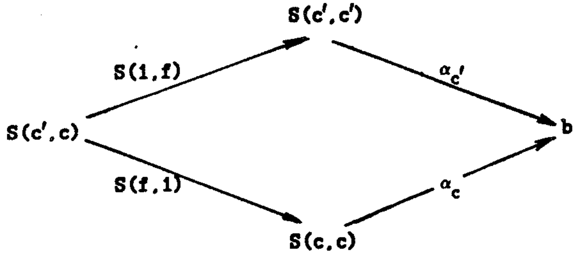
is a function assigning to each object c of $\underline{C}$ a morphism

$$\alpha_c: S(c,c) \rightarrow b$$

in $\underline{B}$ such that for any f: $c\rightarrow c'$ in $\underline{C}$, the following diagram commutes

[8] Here λ denotes the empty string in the set S^* of all strings of sorts.

[9] See [Mac Lane 71], page 215, which explains this concept as a special case of the even more general "dinatural" transformations.



Now take C to be Sign, B to be Set, b={true, false}, put S(Σ',Σ)=Mod(Σ')×Sen(Σ) and let α_Σ be $|=_\Sigma$. Then we get the simplest institution variant, i.e.,

An institution is a pair of functors Mod: $\underline{\text{Sign}}^{op} \to \underline{\text{Set}}$ and Sen: $\underline{\text{Sign}} \to \underline{\text{Set}}$ with an extranatural transformation |=: Mod(_)×Sen(_) $\dot{\to}$ {true, false}.

A particular advantage of the extranatural formulation is that the commutative diamond displays the satisfaction condition in such a direct way. We can also fully capture the content of the more general Definition 1. First, let | _ |: Cat→Cat denote the functor which regards a category C as a *discrete* category |C|, i.e., which discards all non-identity morphisms from C. Next, let 2 denote the category with two objects, 0 and 1, and with just one non-identity morphism, from 0 to 1. Then

Proposition 2: An institution is a pair of functors Mod: $\underline{\text{Sign}}^{op} \to \underline{\text{Cat}}$ and Sen: $\underline{\text{Sign}} \to \underline{\text{Cat}}$ with an extranatural transformation |=: |Mod(_)|×Sen(_) $\dot{\to}$ 2 . □

The reader may verify that this captures institutions with both sentence and model morphisms, automatically giving both the Satisfaction and Soundness Conditions. Thus, all four variants of the institution concept are captured.

2.3 Some Results about Institutions

This subsection summarizes some results from [Goguen & Burstall 85] about institutions. First, some auxiliary concepts are needed.

Definition 3: Let I and I' be institutions. Then:

1. A **theory** in I is a closed class T of Σ-sentences; i.e., if a sentence s is satisfied by every model of all sentences in T, then s lies in T.
2. Let T and T' be theories with signatures Σ and Σ' respectively. Then a **theory morphism** f: T→T' is a signature morphism f: Σ→Σ' such that if s lies in T then f(s) lies in T', where f(s) is Sen(f)(s). This gives rise to a category $\underline{\text{Th}}_I$ of theories over I.
3. Let I and I' be institutions. Then an **institution morphism** Φ: $I \to I'$ consists of
    a. a functor Φ: Sign→Sign',

b. a natural transformation α: Φ;Sen'$\Rightarrow$Sen, that is, a natural family of functors α_Σ: Sen'($\Phi(\Sigma)$)$\rightarrow$Sen(Σ), and

c. 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

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

for any Σ-model m from I and any $\Phi(\Sigma)$-sentence s' from I'.

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

□

Now the results. A key insight, derived from some earlier work in general systems theory [Goguen 71, Goguen & Ginali 78], is that colimits explicate the basic process of putting things (such as theories) together.

1. If Sign has [finite] colimits, then so does the category $\underline{\mathrm{Th}}_I$ of all theories in I.
2. If Φ: $I \rightarrow I'$ is a sound institution morphism with [finitely] cocontinuous signature part, and if Mod and Mod' preserve [finite] colimits, then $\underline{\mathrm{Th}}_\Phi$: $\underline{\mathrm{Th}}_I \rightarrow \underline{\mathrm{Th}}_{I'}$ is [finitely] cocontinuous.
3. If Φ: $I \rightarrow I'$ is sound, then (roughly) a theorem prover for I can be used for I' theories (see [Goguen & Burstall 85] for details).
4. Enriching an institution with data constraints [or hierarchy constraints] yields another institution (see [Goguen & Burstall 85] for details).
5. We can define duplex and multiplex institutions out of two or more given institutions and suitable institution morphisms, to get another institution that combines the given institutions (see [Goguen & Burstall 85] for details).

We are now in a position to give our (somewhat informal) explication of a **logical programming language** as a programming language which has an institution I such that

- its **statements** are sentences in I,
- its **operational semantics** is (a reasonably efficient form of) deduction in I,
- its **mathematical semantics** is given by models in I (preferably initial).

Notice that sentence morphisms are needed here to make sense of the notion of "deduction in I".

3 Charters

It can be a lot of rather of dull work to prove that something really is an institution, amounting to structural induction over the syntax of sentences. Charters attempt to ameliorate this tedium.

The essential idea of an institution is that when we change the signature, the satisfaction relation changes in a smooth way. Now notice that if we have a free algebra on a set of generators, when we change the generators we get a morphism between the free algebras; that is, the free algebra changes smoothly. But with institutions, we are concerned with changing signatures. This vague train of thought leads us to wonder whether we could construct an institution from some situation involving free algebras, or more abstractly, from an adjunction. The former corresponds to "parchments" and is discussed in Section 4; the more abstract approach corresponds to charters. Charters provide a way to get the satisfaction condition automatically; they also provide a nice abstract view of what a *semantic denotation* is. First, the basic concept (without sentence morphisms; see Section 5.1 for these):

Definition 4: A **charter** C consists of

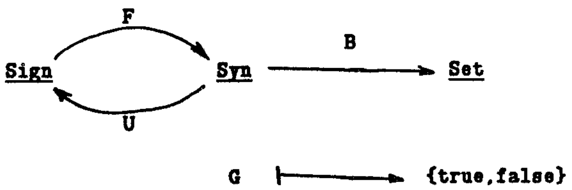
- a category Sign of **signatures**
- an adjunction F-|U: Sign→Syn
- a **ground** object G in |Syn|
- a **base** functor B: Syn→Set

such that

B(G) = {true,false}.

□

The following picture may help in visualizing these relationships:



Section 3.2 gives an example, the equational charter. Roughly, one may think of Syn as a category of "syntactic systems", F as freely constructing such systems over signatures, B as extracting the sentence component from a syntactic system, and G as a "ground" object in

which to interpret other syntactic systems, thus providing models. The following makes all this precise.

3.1 Chartering an Institution

We can construct an institution from a given charter C (i.e., "charter an institution") as follows: Let Σ be a signature in Sign. Then a **Σ-model** is a Sign morphism

$$m: \Sigma \rightarrow U(G)\ ,$$

and the **denotation** morphism for m is the Syn morphism

$$m^{\#}: F(\Sigma) \rightarrow G$$

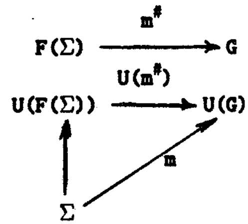
given by the adjunction of F and U. A **Σ-sentence** is an element of

$$Sen(\Sigma) = B(F(\Sigma))\ .$$

Given $e \in Sen(\Sigma)$, $m \in |Mod(\Sigma)|$, we define **satisfaction** by

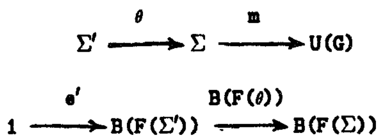
$$m \models e \ \text{ iff } \ B(m^{\#}(e)) = true\ .$$

Let us denote the result of this construction by $\mathcal{I}(C)$. The following diagram may help in visualizing these concepts:



$$B(m^{\#}):\ B(F(\Sigma)) \rightarrow \{true, false\}$$

Let θ: $\Sigma' \rightarrow \Sigma$ be a signature morphism, let m be a Σ-model, and let e' be a Σ'-sentence. Then we define the translation of m by θ, denoted θm, to be the composition, and we define the translation of e' by θ, denoted $\theta e'$, to be $B(F(\theta))(e')$. The diagram below illustrates these definitions:



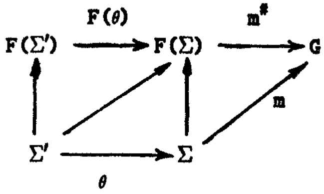
(where 1 denotes a set having a single element, so that a Set morphism from 1 uniquely determines an element).

Lemma 5: Let θ: $\Sigma' \to \Sigma$ be a signature morphism, let m: $\Sigma' \to G$ be a model, and let e' be a Σ-sentence. Then

(1) $F(\theta)m^{\#} = (\theta m)^{\#}$, and

(2) $\theta m \models e'$ iff $m \models \theta e'$.

Proof: (1) follows from the following diagram:



in which U's have been omitted from the top line (i.e., it really should be $U(F(\Sigma'))$ etc.).

The proof of (2) uses (1) as follows: $B((\theta m)^{\#}e') =$ true iff $B(m^{\#}(F(\theta)e')) =$ true. □

Condition (2) above is the "Satisfaction Condition" that is the central property of an institution. Thus, the above gives us

Proposition 6: Given a charter C, the above construction gives an institution $\mathcal{I}(C)$. □

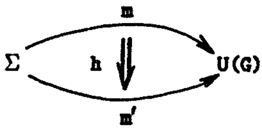
Actually, the above construction works a little more generally. Let C be a charter, and let SubSign be a subcategory of Sign. Then restricting models to be morphisms from signatures in SubSign also gives an institution, denoted $\mathcal{I}_{\underline{SubSign}}(C)$.

An interesting direction for future research concerns the use of colimits in the category of charters (or some related category) in order to "put together" charters, just as previously with theories, and thus to build complex institutions from simpler ones. We have worked out a simple example, the first order predicate calculus with equality, built up in several steps, and we believe that the approach looks promising.

3.1.1 Chartering Model Morphisms

To get model morphisms from chartering, we need some additional structure, namely a 2-category[10] SIGN of **signatures**, having as its underlying horizontal ordinary category Sign. Given m,m': $\Sigma \to U(G)$, a **Σ-morphism** h: $m \to m'$ is a 2-cell in SIGN with source m and target m'. The following picture may help in visualizing this situation:

[10] See [Mac Lane 71], page 44, for the definition of 2-category.



Composition of Σ-homomorphisms is vertical; that is, we take Mod(Σ) to be the category SIGN[Σ,Γ] of 2-cells, where Γ=U(G). Also, given σ: Σ→Σ′ in Sign, we define Mod(σ): SIGN[Σ′,Γ]→SIGN[Σ,Γ] to send m: Σ′→Γ to the horizontal composition σ;m, and to send a 2-cell h: m⇒m′ to the horizontal composition σ;h. The following uses the interchange law for 2-categories:

Proposition 7: Mod as defined above is a functor Sign → Cat^{op}. □

One might think it unusual for Sign to have the necessary additional structure of a 2-category. However, there is a simple trick that covers many cases of interest: we need only give each Sign(Σ,U(G)) the structure of a category and use the arrows in them as 2-cells, with no other non-identity 2-cells, to get a suitable 2-category SIGN. This is illustrated in Section 3.2 below for the equational charter.

3.2 The Equational Charter

This subsection outlines the many-sorted equational charter, yielding the many-sorted equational institution under the construction of Section 3.1.

- Sign is the category SigAlg of Section 2.1.
- Syn is a category of models for notions of term and equation, for various signatures; in particular, the free algebras, F(Σ), can be seen as triples ⟨Σ,T,E⟩, where T is a sorted family of Σ-terms and E is the set of all Σ-equations, plus some operations. More precisely now, let us fix an infinite set $\mathcal{X}$ of "variable" symbols. Given a signature Σ, let S be its sort set, and let $\mathcal{V}=\{\mathcal{X}\xrightarrow[\mathrm{fin}]{}S\}$, where $\{A\xrightarrow[\mathrm{fin}]{}B\}$ denotes the set of *partial* functions from A to B that are only defined on a *finite* number of elements of A; elements of $\mathcal{V}$ are in effect "finite-sorted variable sets from $\mathcal{X}$". Letting $S^+=\{*\}\cup(\mathcal{V}\times S)$, we define Σ^+ to be the S^+-sorted signature with
  - o $\Sigma^+_{\lambda,(X,s)}=\{x\in\mathcal{X} \mid X(x)=s\}$
  - o $\Sigma^+_{(X,s1)\ldots(X,sn),(X,s)}=\Sigma_{s1\ldots sn,s}$
  - o $\Sigma^+_{(X,s)(X,s),*}=\{=_{(X,s)}\}$ and
  - o all other components of Σ are empty,

  where $X\in\mathcal{V}$, s,s1,...,sn∈S, and $w\in S^{+*}$. Also, given σ: Σ→Σ′, let σ^+: $\Sigma^+\to\Sigma'^+$ be the obvious extension of σ making $(\_)^+$ a functor. Then the objects of Syn are pairs $\langle\Sigma,A\rangle$ where A is a Σ^+-algebra, and the morphisms $\langle\Sigma,A\rangle\rightarrow\langle\Sigma',A'\rangle$ in Syn are pairs $\langle\sigma,h\rangle$ where $\sigma\colon\Sigma\rightarrow\Sigma'$, $h\colon A\rightarrow A'|_{\sigma^+}$ and $A'|_{\sigma^+}$ is A′ regarded as a Σ^+-algebra by using σ^+. Identity and composition in Syn are the obvious choices (see also Section 4).

- U: Syn→Sign sends $\langle\Sigma,A\rangle$ to Σ and sends $\langle\sigma,h\rangle$ to σ.
- Define the functor F to send Σ to $\langle\Sigma,T_{\Sigma^+}\rangle$ where T_{Σ^+} denotes an initial Σ^+-algebra, such as a term algebra.
- Let us now define a "procrustean ground signature" Γ having sort set S=|Set| for some category Set of sets, and for w∈S* and s∈S having $\Gamma_{w,s}=[\Pi w\rightarrow s]$, the set of all functions from the product of the sets in w to the set s. Given $X\in\mathcal{V}$ (i.e., X is a finite partial function from $\mathcal{X}$ to |Set|) let $Env(X)=\Pi\{X(x) \mid X \text{ is defined at } x\}$[11]. We now define a Γ^+-algebra $\mathbb{G}$ as follows:
  - o $\mathbb{G}_*=\{true,false\}$;
  - o $\mathbb{G}_{\langle X,s\rangle}=[Env(X)\rightarrow s]$;
  - o for x in $\Gamma^+_{\lambda,\langle X,s\rangle}$ (i.e., for x such that X(x)=s), $\mathbb{G}_x$ denotes the function in $[Env(X)\rightarrow s]$ sending $\underline{a}$ in Env(X) to $\underline{a}_x$ in s;
  - o for σ in $\Gamma^+_{\langle X,s1\rangle\ldots\langle X,sn\rangle,\langle X,s\rangle}$ let $\mathbb{G}_\sigma$ send $\langle f1,\ldots,fn\rangle$ in $\Pi^n_{i=1}[Env(X)\rightarrow si]$ to $\langle f1,\ldots,fn\rangle;\sigma$ in $[Env(X)\rightarrow s]$, noting that $\sigma\colon s1\times s2\ldots\times sn\rightarrow s$; and
  - o $\mathbb{G}_{=_{\langle X,s\rangle}}\colon [Env(X)\rightarrow s]^2 \rightarrow \{true,false\}$ is defined by $f=_{\langle X,s\rangle}g$ is true iff f=g (as functions from Env(X) to s).

  Now let $G=\langle\Gamma,\mathbb{G}\rangle$.
- Finally, B: Syn→Set is the forgetful functor extracting the elements of the equation sort, i.e., sending $\langle\Sigma,A\rangle$ to A_*.

In order to show that this is a charter, we have to verify that F is really left adjoint to U and that B is really a functor. These results are not difficult, but we will see in Section 4, Lemmas 10 and 11, that they follow automatically from the nature of this charter, more precisely, from the fact that it arises from a parchment.

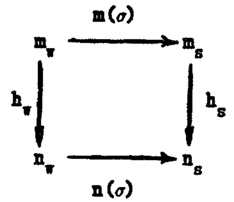
Let us briefly consider some other examples. For predicate calculus, Sign would have signatures giving function and relation symbols. For order-sorted equational logic, it would have order-sorted signatures, which provide an ordering relation on the sort set. Doing equational logic for continuous algebras would leave Sign as SigAlg, but Σ^+ could add '=', '≤' and an infinite union; the ground signature Γ would have as sorts complete partial

---

[11] Env is chosen to suggest "environment," as in denotational semantics; an element of Env(X) maps a variable x to a value in X(x), when this is defined.

orders, and now $\Gamma_{w,s}$ would be the set of all *continuous* functions. Notice that the partial order structure would not affect the sentences, which would still be elements of a term algebra; contrast this with denotational semantics where the domain structure creeps, unnecessarily one might think, into the syntax.

We now give Sign a 2-category structure which will yield the expected many-sorted algebra homomorphisms, following the method of Section 3.1.1. Assume that we are given two Σ-models, i.e., two signature morphisms m,n: $\Sigma \rightarrow \Gamma$, and let S be the sort set of Σ. Then let us write m_s for $m_1(s)$, where m_1 is the sort component of m and $s \in S$ (similarly for n_s). Now define a 2-cell h: $m \Rightarrow n$ to be a family $\langle h_s \mid s \in S \rangle$ such that the following diagram commutes for each σ in $\Sigma_{w,s}$



where $m_w = \Pi_{i=1}^{k} m_{si}$ when w=s1...sk and $m_w = 1$, some singleton set, when k=0 (i.e., when $w = \lambda$). This permits us to regard any homset of the form Sign(Σ, Γ) as a category; we then say that the only other 2-cells in SIGN are identities. This gives us a 2-category structure on Sign and thus a notion of Σ-homomorphism. The reader may check the following:

Proposition 8: The homset Sign(Σ, Γ) with morphisms 2-cells of the category SIGN as described above, is a category isomorphic to the usual category $\underline{Alg}_{\Sigma}$ of many-sorted Σ-algebras and Σ-homomorphisms. □

4 Parchments

Although we get satisfaction automatically by chartering an institution, it is still some trouble to describe Syn and to construct the adjunction. Parchments will give us these for "free" also.

Let us briefly review the setup for an initial algebra semantics of a formal language (either a programming language or a logical language) [Goguen 74, Goguen, Thatcher, Wagner & Wright 77]. There is a signature whose sorts name the various classes of syntactic entities (i.e., the "phrases"), and whose operation symbols name the various syntactic constructions, such as building a term from a constant, or from a function symbol and a tuple of terms; the language is then the initial algebra on this "syntactic signature". Given a particular "ground" algebra, each sort is interpreted as whatever that syntactic class denotes, and each operator is interpreted as a semantic function corresponding to a syntactic construction. The denotation function is the unique homomorphism from the intitial algebra (the language) to the ground algebra (of meanings).

In our application to logical languages, we have a syntactic signature whose sorts denote things like terms and sentences, and whose operations construct these. Since this syntactic signature is just a many-sorted signature of the usual kind, it is an object of SigAlg, the usual category of signatures for algebras; since the syntactic signature is constructed from a given signature of operation symbols in a uniform way, we expect to have a functor from Sign to SigAlg, say Lang: Sign→SigAlg. Most familiar institutions can be treated in this way; for example, constructions that can be viewed this way are given in [Goguen & Burstall 85] for many-sorted first order logic, equational logic, and Horn clause logic; Section 4.1 below gives details for equational logic. The basic idea of parchments is to use initial algebra semantics, parameterized by signature with a functor Lang as above, to define the syntax of sentences; the adjoint needed in a charter expresses the initiality of this construction. This is a formalization of the insight, hardly new in itself, that the syntax of logic is an initial algebra; [Lyndon 66] is a rather early and quite charming reference; there is actually a considerable literature, including several book length developments, on the algebraicization of logic.

Definition 9: A **parchment** $\mathcal{P}$ consists of:

- a functor Lang: Sign→SigAlg
- a signature Γ in |Sign|
- a **ground** algebra $\mathfrak{G}$ in |Alg(Lang(Γ))| and
- an element $*$ in sorts(Lang(Σ)) for each Σ in |Sign|

such that

- $\mathfrak{G}_* = \{\text{true,false}\}$ and
- Lang(σ)($*$) = $*$ for each morphism σ: $\Sigma \to \Sigma'$ in Sign.

□

The intuition is that Lang(Σ) gives syntax for constructing Σ-sentences, which lie in $T_{\text{Lang}(\Sigma),*}$ and are the "interesting" part of the free algebra over Lang(Σ). Moreover, $\mathfrak{G}$ is a semantic "ground" for interpretation. (See Section 5.1 for the additions needed to get sentence morphisms.)

There is a recipe for writing a charter on a given parchment:

- First, define the category Syn as follows[12]:
  - its objects are pairs $\langle\Sigma,A\rangle$, where A is in Alg(Lang(Σ)); and
  - its morphisms from $\langle\Sigma,A\rangle$ to $\langle\Sigma',A'\rangle$ are pairs $\langle\sigma,h\rangle$ where $\sigma: \Sigma\to\Sigma'$ is a signature morphism and $h: A\to A'^{Lang(\sigma)}$ is a Lang(Σ)-homomorphism.
  - If $\langle\sigma,h\rangle: \langle\Sigma,A\rangle \to \langle\Sigma',A'\rangle$ and $\langle\sigma',h'\rangle: \langle\Sigma',A'\rangle \to \langle\Sigma'',A''\rangle$, then we define the composition $\langle\sigma,h\rangle;\langle\sigma',h'\rangle$ to be $\langle\sigma;\sigma',h;h'^{Lang(\sigma)}\rangle$. Also, define $1_{\langle\Sigma,A\rangle}=\langle 1_\Sigma,1_A\rangle$.
- Next, let B: Syn$\to$Set send $\langle\Sigma,A\rangle$ to A_* and send $\langle\sigma,h\rangle$ to $h_*: A_*\to A'_*$; the proof that this really is a functor is given in Lemma 10 below.
- Define U: Syn$\to$Sign to send $\langle\Sigma,A\rangle$ to Σ and to send $\langle\sigma,h\rangle$ to σ.
- Then U has a left adjoint F: Sign$\to$Syn sending Σ to the pair $\langle\Sigma,T_{Lang(\Sigma)}\rangle$; the proof that this is an adjoint is given in Lemma 11 below.
- Finally, define $G=\langle\Gamma,\mathfrak{C}\rangle$.

Lemma 10: B as defined above is a functor.

Proof: Suppose that $\langle\sigma,h\rangle: \langle\Sigma,A\rangle \to \langle\Sigma',A'\rangle$ and $\langle\sigma',h'\rangle: \langle\Sigma',A'\rangle \to \langle\Sigma'',A''\rangle$. Then

$$B(\langle\sigma,h\rangle);B(\langle\sigma',h'\rangle): \langle\Sigma,A\rangle \to \langle\Sigma'',A''\rangle$$

is

$$h_*;h'_*:A_* \to A''_*$$

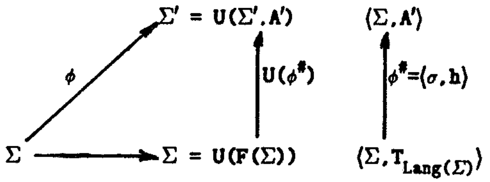
while the composition $\langle\sigma,h\rangle;\langle\sigma',h'\rangle$ is $\langle\sigma;\sigma',h;h'^{Lang(\sigma)}\rangle$, and taking B of this yields

$$(h;h'^{Lang(\sigma)})_* = h_*;h'_*$$

as desired, since $h'^{Lang(\sigma)}_*=h'_*$ since Lang(σ)(*)=*. Also, of course, $B(1_{\langle\Sigma,A\rangle})=B(1_\Sigma,1_A)=1_{A_*}$. □

Lemma 11: F as defined above is left adjoint to U.

Proof: It suffices to show that F(Σ) is free with respect to U; then functoriality and adjointness follow automatically. Thus, assuming we are given Σ, $\langle\Sigma',A'\rangle$, and $\phi: \Sigma\to\Sigma'$, we want to show that there is a unique $\phi^\#=\langle\sigma,h\rangle: F(\Sigma)\to\langle\Sigma',A'\rangle$ such that the following diagram commutes:



---

[12]Syn is actually the "flattening" of an indexed category, the functor Syn: Sign$\to$Cat^{op} which assigns the category Alg(Lang(Σ)) to each Σ in Sign.

Taking the unit η_Σ to be 1_Σ, commutativity gives $\sigma=\phi$ since $U(\phi^{\#})=\sigma$, and initiality of $T_{Lang(\Sigma)}$ gives that there is a unique h: $T_{Lang(\Sigma)} \to A'^{Lang(\sigma)}$. □

Thus, we have

Proposition 12: The above recipe yields a charter from a parchment $\mathcal{P}$, and thus an institution $\mathcal{I}(\mathcal{P})$. □

Notice that we can get model morphisms in the institution of a parchment from a 2-category structure SIGN on its category Sign of signatures, since its charter inherits this structure, and we can then use the method already given for chartering model morphisms.

We can now give the promised explication of finitude for specifications: A **specification** should involve only a finite set of Σ-sentences in an institution that can be chartered by a parchment. For example, a set of Σ-equations can be considered a specification in this sense, since equational logic is a parchment chartered institution by the following subsection. Similarly, a set of first order sentences over given signatures Σ and Π of function and relation symbols (respectively) is also a specification, since (many-sorted) first order logic is a parchment chartered institution.

4.1 The Equational Parchment

We now give the ingredients of the equational parchment:

- Sign is SigAlg, as in Section 3.2.
- Letting $\mathcal{X}$ be a fixed infinite set of variable symbols, define Lang to be $(\_)^+$: SigAlg→SigAlg, sending Σ to Σ^+ as in Section 3.2, where sorts(Σ^+)={*}∪{(X,s) | X∈{$\mathcal{X} \xrightarrow{fin} S$}, s∈S} when S=sorts(Σ). This functor Lang satisfies Lang(σ)(*)=* for any signature morphism σ: Σ→Σ'. Finally, let Γ and the Γ^+-algebra ₵ be as in Section 3.2. This gives a parchment.

Just for fun, let's see what satisfaction is in its institution. First, notice that Sen(Σ)=B(F(Σ))=$T_{Lang(\Sigma),*}$ as desired. Now letting m: Σ→Γ and e∈$T_{Lang(\Sigma),*}$ note that $m|=_\Sigma e$ iff B($m^{\#}_*$(e))=true iff $m^{\#}_*$(e)=true. (Note that $m^{\#}_*$: $T_{\Sigma^+,*} \to ₵^{m^+}_*$, i.e., that $m^{\#}_*$: Sen(Σ)→{true,false} as needed.) Say e is ($t1 =_{(X,s)} t2$). Then $m^{\#}_*(t1 =_{(X,s)} t2)$=true iff indeed $m^{\#}_{(X,s)}(t1) = m^{\#}_{(X,s)}(t2)$ as functions from Env(X) to s. So this really is the equational institution.

Thus, in summary, to found an institution, define its category Sign of signatures, a functor Lang: Sign→SigAlg, a signature Γ, and a Lang(Γ)-algebra $\mathcal{G}$; the construction of the ground algebra $\mathcal{G}$ is likely to be the hardest part of this, but satisfaction is guaranteed. As noted before, the insight that the language of logic is an initial algebra is hardly new; but we believe our formalization of this insight is new and also has some interesting applications. The reader may wish to try founding some other instutitions on parchments, such as first order logic, order-sorted algebra, or continuous algebra.

5 Generalized Institutions

A broad range of applications for an institution-like concept have been suggested by [Mayoh 85], including database query systems, knowledge representation systems and programming languages. The intuition is simply that broadening the notion of "truth value"[13] allows more general sentences and models; for example, given a sentence s in a database query language and a model D which is a suitable database, the generalized "satisfaction" relation has as its value the response to the query s for the database D. Proposition 2 and the work of [Mayoh 85] both suggest generalizing the concept of institution by replacing the category 2 in Proposition 2 by an arbitrary **value category** V. This not only generalizes Mayoh's concept, but also expresses it more elegantly in categorical language and moreover patches what seems a bug in Mayoh's original formulation. (Recall that |_|: Cat→Cat denotes the functor which regards a category C as a *discrete* category |C|, i.e., it discards all non-identity morphisms from C.)

Definition 13: Let V be a category. Then a (generalized) **V-institution** is a pair of functors Mod: $\underline{\text{Sign}}^{op}$→Cat and Sen: Sign→Cat with an extranatural transformation $\models$: $|\text{Mod}(\_)| \times \text{Sen}(\_) \dashrightarrow \underline{V}$. □

Institutions are the special case of generalized institutions where V=2. Mayoh's galleries correspond to the special case where the sentence categories are all discrete and V is the category of sets. Mayoh calls institutions in our original sense "logical galleries". But there seems to be an unfortunate bug in Mayoh's formulation: his model morphisms must be sentence-truth-preserving, and this seems inadequate even for the equational institution, since it would allow little more than quotient homomorphisms. Correcting this bug was the reason for the discretization functor |_| above. On the other hand, sentence morphisms (which our generalization admits but Mayoh's galleries do not) work out beautifully, since they are *supposed* to be truth-preserving. Although Mayoh suggests some nice examples in his

---

[13] This step may remind the reader of fuzzy sets.

framework, he does not make a convincing case that the framework actually helps in treating the examples, and he does not actually prove that the satisfaction axiom holds. Mayoh's work suggests what seems an exciting approach to the semantics of database systems etc., but more study seems to be needed. The framework of generalized institutions, with its additional feature of sentence morphisms, suggests some additional topics for further study; one which seems especially intriguing is to consider program transformations as sentence morphisms.

Although the wedge formulation seems more elegant, it may be also of interest to give our generalization in a form that is closer to Mayoh's formulation.

Definition 14: A **generalized V-room** consists of categories M and S, and a functor r: |M|→[S→V], where V is a **value** category, M is a **model** category, S is a **sentence** category, and [S→V] denotes the functor category.

Let r and r′ be generalized V-rooms. Then a **generalized V-room morphism** from r to r′ is a pair of functors f: M′→M, g: S→S′ such that the following diagram commutes:

$$\begin{array}{ccc} |M| & \xrightarrow{\;r\;} & [S\to V] \\ {\scriptstyle |f|}\uparrow & & \uparrow{\scriptstyle [g\to V]} \\ |M'| & \xrightarrow[\;r'\;]{} & [S'\to V] \end{array}$$

Let Room(V) denote the category of generalized V-rooms and generalized V-room morphisms. Then a **generalized institution**[14] is a functor Sign → Room(V). □

Proposition 15: The above definition of generalized institution agrees with that given in Definition 13. □

We note that Mayoh calls an object of the functor category [Sen(Σ)→Set] a "data type", and calls such an assignment of values to sentences a "realizable data type" if it arises from a model. This seems overly general, since only realizable data types are of real interest.

There is also a very nice formulation of generalized institutions as the objects of a diagram

[14] An earlier version of this paper called this concept a **society** and used the word **clan** for generalized room; the current names were chosen to emphasize the similarity to prior work. But doesn't it sound nice to say that "a society is a functor from signatures to clans"?

category; in fact, one can define the whole category of institutions in just eleven symbols[15]!

Proposition 16: The category of generalized institutions is

$\underline{D}(|\_|^{op}/\underline{V}^{-})$,

where $|\_|$ is the discretization functor, so that $|\_|^{op}$: $\underline{Cat}^{op} \to \underline{Cat}^{op}$, where $\underline{V}^{-}$ denotes the functor $\underline{Cat} \to \underline{Cat}^{op}$ assigning the functor category $[\underline{A} \to \underline{V}]$ to the category $\underline{A}$, and where $\underline{D}(\underline{C})$ is the **diagram category**[16], whose objects are functors F: $\underline{A} \to \underline{C}$ for some category $\underline{A}$, and whose morphisms from F to G: $\underline{B} \to \underline{C}$ are pairs $\langle \Phi, \phi \rangle$ where Φ: $\underline{A} \to \underline{B}$ and ϕ: $\Phi;G \Rightarrow F$ is a natural transformation. □

To see why Proposition 16 is true, first notice that the category of $\underline{V}$-rooms is the comma category $(|\_|^{op}/\underline{V}^{-})$; for, the objects of the comma category are triples $\langle \underline{M}$, r: $|\underline{M}| \to [\underline{S} \to \underline{S}]$, $\underline{S} \rangle$ and its morphisms are pairs $\langle$f: $\underline{M}' \to \underline{M}$, g: $\underline{S} \to \underline{S}' \rangle$ such that the diagram in Definition 14 commutes. Thus, an object of the diagram category $\underline{D}(|\_|^{op}/\underline{V}^{-})$ is a functor with target $\underline{Room}(\underline{V})$, i.e., a $\underline{V}$-institution. Given I: $\underline{Sign} \to \underline{Room}(\underline{V})$ and I': $\underline{Sign}' \to \underline{Room}(\underline{V})$, a morphism $I \to I'$ is a pair $\langle \Phi, \phi \rangle$ with Φ: $\underline{Sign} \to \underline{Sign}'$ and ϕ: $\Gamma;I' \Rightarrow I$ is a natural transformation. The natural way that the morphisms in this formulation arise from the general $\underline{D}$ and "comma" constructions provides additional motivation for the definition of institution morphism that we have given. This argument relies upon the following, whose proof we leave to the reader.

Fact 17: The morphisms in the category of Proposition 16 agree with institution morphisms of Definition 3 when $\underline{V} = \underline{2}$. □

The formulation of Definition 16 also gives an elegant proof of the following

Proposition 18: The category of institutions is cocomplete. □

The proof uses well-known cocompleteness results for the diagram and comma category constructions, plus cocompleteness of $\underline{Cat}$. Of course, this argument also gives cocompleteness of the category of generalized institutions. This result is important because it allows us to put old institutions together to make new institutions; in fact, we can proceed just as we would with putting theories together with a specification language, using colimits to achieve parameterization, as in the specification language Clear, and we could even use a Clear-like syntax.

---

[15] See [Mac Lane 71] page 47 or [Goguen & Burstall 84] for the definition of comma categories.

[16] [Mac Lane 71] page 111 calls this is a "super comma category".

5.1 Generalized Charters and Parchments

There are also generalized notions corresponding to charter and parchment, called **generalized charter** and **generalized parchment**. The definitions are remarkably simple modifications of the original definitions. At the same time, we also handle sentence morphisms. A generalized charter has B: Syn→Cat instead of B: Syn→Set, and the recipe for constructing a generalized institution defines the value category V to be B(G). This seems both simpler and more general than the original charter concept; together with the following result, the naturalness of both generalized charters and institutions is reinforced.
Proposition 19: A generalized charter yields a generalized institution under the recipe of Section 3, with V=B(G) and with $m \models e$ defined to be $B(m^{\#}(e))$. □

Let us now consider parchments. The generalized definition requires a sort denoted "↦" in each Lang(Σ) in additional to the sort denoted *, plus two operator symbols $@_i$ in $\mathrm{Lang}(\Sigma)_{\mapsto,*}$ for i=0,1 for each Σ, such that Lang(σ)(*)=*, Lang(σ)(↦)=↦, and $\mathrm{Lang}(\sigma)(@_i)=@_i$ for i=0,1 and each morphism σ in Sign. Then we have
Proposition 20: A generalized parchment yields a generalized charter by following the recipe of Section 4, modified by defining B: Syn→Cat to send ⟨Σ,A⟩ to $\mathrm{Pa}(A^{gr})$, where A^{gr} is the Lang(Σ)-algebra A regarded as a graph by using *, ↦, $@_0$ and $@_1$, and where Pa(G) is the category of paths in a graph G. □

Finally, we note that the same device used to get model morphisms for ordinary charters and parchments extends to the generalized case, i.e., we need only assume a 2-category SIGN with underlying ordinary category Sign.

All this simplicity underscores the fact that the "generalized" concepts are a very natural extension of the original concepts of institution, charter and parchment.

6 Conclusions

We have given a number of equivalent formulations of the institution concept, and have argued that institutions are a useful abstraction in theoretical computer science. In particular, we have recalled that institutions have many pleasant properties, many important instances, and some interesting applications; moreover, this paper has introduced institutions with sentence morphisms and used them to clarify the notion of "logical" programming language. In addition, we have argued that notions of specification which involve essentially infinite sentences are examples of unsuitable abstraction, and we have clarified the sense of finitude involved by using parchments. Mayoh's galleries suggest exciting further applications, but may not be quite general enough, and do not seem to give the right model morphisms. Galleries and the extranatural transformation formulation of institution, motivate our concept of generalized institution. Finally, we have introduced generalized charters and parchments.

References

[Abrial, Schuman & Meyer 79]
Abrial, J. R., S. A. Schuman and B. Meyer.
Specification Language (draft).
1979.
Cambridge University.

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

[Beierle & Voss 85]
Beierle, Christoph and Angelika Voss.
Implementation Specifications.
Technical Report 147/85, Universität Kaiserslautern, 1985.

[Bjorner & Jones 78]
Bjorner, Dines and Cliff Jones.
The Vienna Development Method.
Lecture Notes in Computer Science 61, 1978.

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

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

[Burstall & Goguen 81]
Burstall, Rod and Joseph Goguen.
An Informal Introduction to Specifications using Clear.
In Robert Boyer and J Moore (editors), *The Correctness Problem in Computer Science,* pages 185-213. Academic Press, 1981.
Reprinted in *Software Specification Techniques,* edited by N. Gehani and A. D. McGettrick, Addison-Wesley, 1985, pages 363-390.

[Futatsugi, Goguen, Jouannaud & Meseguer 85]
Futatsugi, Kokichi, Joseph Goguen, Jean-Pierre Jouannaud and José Meseguer.
Principles of OBJ2.
In *Proceedings, Symposium on Principles of Programming Languages*, pages 52-66. Association for Computing Machinery, 1985.

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

[Goguen 74] Goguen, Joseph.
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, Volume 25, Springer-Verlag, 1975, pages 151-163.

[Goguen 86] Goguen, Joseph.
Reusing and Interconnecting Software Components.
Computer 19(2):16-28, February, 1986.

[Goguen & Burstall 84]
Goguen, Joseph and Rod Burstall.
Some Fundamental Algebraic Tools for the Semantics of Computation, Part 1: Comma Categories, Colimits, Signatures and Theories.
Theoretical Computer Science 31(2):175-209, 1984.

[Goguen & Burstall 85]
Goguen, Joseph and Rod Burstall.
Institutions: Abstract Model Theory for Computer Science.
Technical Report CSLI-85-30, Center for the Study of Language and Information, Stanford University, 1985.
Also submitted for publication; a preliminary version appears in *Proceedings, Logics of Programming Workshop*, edited by Edward Clarke and Dexter Kozen, volume 164, Springer-Verlag Lecture Notes in Computer Science, pages 221-256, 1984.

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

[Goguen & Meseguer 85]
Goguen, Joseph and José Meseguer.
Completeness of Many-sorted Equational Logic.
Houston Journal of Mathematics 11(3):307-334, 1985.
Preliminary versions have appeared in: *SIGPLAN Notices*, July 1981, Volume 16, Number 7, pages 24-37, and January 1982, Volume 17, Number 1, pages 9-17; SRI Technical Report CSL-135, May 1982; and Technical Report CSLI-84-15, Center for the Study of Language and Information, Stanford University, September 1984.

[Goguen & Meseguer 86a]
Goguen, Joseph and José Meseguer.
Eqlog: Equality, Types, and Generic Modules for Logic Programming.
In Douglas DeGroot and Gary Lindstrom (editors), *Functional and Logic Programming*, pages 295-363. Prentice-Hall, 1986.
An earlier version appears in the *Journal of Logic Programming*, volume 1, number 2, pages 179-210, September 1984.

[Goguen & Meseguer 86b]
Goguen, Joseph and José Meseguer.
Object-Oriented Programming as Reflective Equational Programming.
In preparation.
1986

[Goguen, Thatcher, Wagner & Wright 77]
Goguen, Joseph, James Thatcher, Eric Wagner and Jesse Wright.
Initial Algebra Semantics and Continuous Algebras.
Journal of the Association for Computing Machinery 24(1), January, 1977.

[Lloyd 84] Lloyd, J. W.
Foundations of Logic Programming.
Springer-Verlag, 1984.

[Lyndon 66] Lyndon, Roger C.
Mathematical Studies. Volume 6: *Notes on Logic.*
Van Nostrand, 1966.

[Mac Lane 71] Mac Lane, Saunders.
Categories for the Working Mathematician.
Springer-Verlag, 1971.

[Mayoh 85] Mayoh, Brian.
Galleries and Institutions.
Technical Report DAIMI PB-191, Aarhus University, 1985.
This contains a number of reports, some of which have been presented at various conferences.

[Reichel 84] Reichel, Horst.
Structural Induction on Partial Algebras.
Akademie-Verlag, 1984.

[Sanella & Tarlecki 85a]
Sanella, Donald and Andrzej Tarlecki.
On Observational Equivalence and Algebraic Specification.
In *Lecture Notes in Computer Science.* Volume 185: *Mathematical Foundations of Software Development, volume 1: Proceedings of the Colloquium on Trees in Algebra and Programming,* pages 308-322. Springer-Verlag, 1985.
Also appeared as University of Edinburgh, Department of Computer Science Technical Report CSR-172-84.

[Sanella & Tarlecki 85b]
Sanella, Donald and Andrzej Tarlecki.
Building Specifications in an Arbitrary Institution.
In Giles Kahn, David MacQueen and Gordon Plotkin (editor), *Lecture Notes in Computer Science.* Volume 173: *Proceedings, International Symposium on the Semantics of Data Types,* pages 337-356. Springer-Verlag, 1985.
Also appeared as Internal Report CSR-184-85, University of Edinburgh, Department of Computer Science.

[Tarlecki 84] Tarlecki, Andrzej.
Free Constructions in Algebraic Institutions.
In *Lecture Notes in Computer Science.* Volume 176: *Proceedings, Int. Symp. Math. Foundations of Computer Science,* pages 526-534. Springer-Verlag, 1984.
Extended version, University of Edinburgh Computer Science Department Report CSR-149-83, and revised version 'On the existence of Free Models in Abstract Algebraic Institutions', September 1984.

[van Emden & Kowalski 76]
van Emden, Maartin H. and Robert Kowalski.
The Semantics of Predicate Logic as a Programming Language.
Journal of the Association for Computing Machinery 23(4):733-742, 1976.