Towards a Semantic Basis for Rosetta

Catherine Menon, Charles Lakos
Department of Computer Science
Adelaide University
{menon, Charles.Lakos}@cs.adelaide.edu.au

Cindy Kong
Information and Telecommunication Technology Center
University of Kansas
kong@ittc.ku.edu

Abstract

Rosetta is a specification language for designing hardware and software systems with a view to being able to consider multiple system perspectives concurrently (such as functional correctness, performance constraints, and physical constraints). It also provides the capability to integrate components from heterogeneous domains, such as software and reconfigurable systems and digital and analog hardware.

This paper explores a semantic framework for Rosetta in terms of category theory. We outline how a specification can be equated with a theory, and the different components which satisfy a specification can be equated with the algebras which satisfy this theory. From this, we define a category consisting of all legal specifications in a given language, and then show how the notions of extension and interaction of specifications can be modeled by means of colimits within this category. This requires a consideration of the different notions of equivalence of specifications, including behavioural equivalence, axiom abstraction and signature abstraction. Finally, we briefly outline how information hiding might be included in the construction of objects within the category of theories.

While the motivation for this work is the semantics of Rosetta, the approach adopted can be applied to other specification languages which support multifaceted, heterogeneous specifications.

Keywords: semantics, modelling, specification, category theory, theories, institutions, Rosetta

1 Introduction

In this paper we briefly introduce Rosetta (Alexander, Kong, Ashenden, Menon, & Barton 2000), a requirements and modelling language currently under development at the University of Kansas. Rosetta is to be used to specify both hardware and software systems, with a primary objective to specify mixed-signal systems. These are systems which incorporate both digital and analog components, and are prevalent in the growing number of system-on-a-chip designs. One of the major points of interest of Rosetta is its support for interactions, by which components from different semantic environments can share information. One example of this is interfacing a digital and an analog component together. To do this, we must provide a semantics that allows transfer of information from the digital to the analog domain, yet still allows each component to retain those design abstractions associated with its particular environment.

We propose a semantic basis for Rosetta in a category theory framework. We do this in the interest of achieving regularity of definitions and theories, of dealing with abstraction and representation independence, and of unifying the different domains addressed by Rosetta (Goguen 1991). The proposed semantics draws on the semantics of Clear (Goguen & Burstall 1991). However, we extend this with the additional capabilities which Rosetta provides. These include aspects such as information hiding and the interactions mentioned earlier, as well as an emphasis placed on simulation in addition to specification. In effect, Rosetta seeks to combine some of the capabilities of Clear, VHDL-AMS (Ashenden, Peterson & Teegarden 2003), and Ptolemy (Davis 1999) and as such incorporates a number of different semantic concepts. It is the aim of this paper to project the elements of Rosetta into the category theory domain, and examine which concepts in category theory have a parallel in Rosetta, and vice versa.

2 Elements of Rosetta

In Rosetta, the primary unit of specification is a facet, which defines a single aspect of a component or system from a particular perspective (Alexander et al. 2000). For example, if specifying an alarm clock, one facet might specify how the alarm clock settings might be changed, one might specify the storage of those settings, and one might specify the running of the clock. Thus, facets may correspond to actual tangible components in a system, abstract design entities, or any other subsystem depending on the requirements of the specification. A facet consists of a collection of variables and functions, and a series of equations which constrain the values of these variables. In Rosetta, the most common way facets are used is to specify or describe the behaviour of one component of a system. However, there are certain
facets whose purpose is simply to provide a vocabulary of definitions, functions, variables, equations and datatypes for other facets to use. These are known in Rosetta as domains. Rosetta incorporates an inheritance mechanism known as extension, whereby a facet which extends a given domain inherits all declarations and axioms from that domain. It is worth noting that the distinction between facets and domains exists primarily for historical reasons, and reasons of convenience. Thus, all domains in Rosetta are in fact facets, and can extend other domains in the same way ordinary facets do.

2.1 Example

The following code illustrates the extension mechanism by presenting the state-based domain, and a facet which extends this. For convenience, the domain is simplified and includes fewer elements than the current Rosetta state-based domain.

domain state-based :: logic is
<table>
<thead>
<tr>
<th>State:</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>-&gt;</td>
</tr>
<tr>
<td>next State</td>
<td>-&gt;</td>
</tr>
<tr>
<td>end state-based</td>
<td></td>
</tr>
</tbody>
</table>

facet increment :: state-based is
private x: bit;
private s: State;

begin
L1: xinit=0;
L2: xnext(s) = (x+x) + 1;
end increment

The header of the facet or domain is given by facet (facet-name) and gives both the name of the facet and the parameters, if any. (Note, there is no semantic difference between parameters and exported variables.) All variables and constants, including those variables of type function, are declared between the facet header and the keyword begin. The name of the domain which is being extended, should there be one, occurs in the header and implies that all that is declared in this domain is visible in these facets. Finally, a collection of labelled terms is included, which act as constraints on the declared variables.

With reference to the example above, the state-based domain extends the logic domain, where such things as bit and + are declared. The type State and functions initial and next are inherited by facet increment, which extends the state-based domain. Any constraints in state-based must also hold in increment.

2.2 Domain Hierarchy

Because Rosetta aims to provide a requirements and simulation framework for heterogeneous systems, it must take into account the different methods of interpreting information and especially interpreting how data values change relative to time, space, or other data. For example, a clock simulation is state-based and thus requires the notions of state, initial state, next-state, and that each change in the value of a variable is related to a state-change. In contrast, a system for modelling heat-flow requires the notion of a variable that changes value smoothly, with respect to time, rather than discretely with respect to change of state. Furthermore, this change may not be specified directly, but rather in the form of a differential equation. It is clear that these two systems do share some elements, but that they also use different and very specialized vocabularies. This is captured by an extension hierarchy of the domains. At the top, the most general domain is the logic domain, which includes those axioms of first-order logic, as well as the definitions of the integers. Extending this in different ways gives us the state-based and signal-based domains, which each require their own definitions and constraints as well as those declared in logic. Part of the extension hierarchy is shown in figure 1.

2.3 Interactions

One of the strengths of Rosetta, as mentioned earlier, is its support for facet interactions. In the most basic form, this simply refers to two facets sharing data or constraints. There are a number of common interactions which have been found to be useful, including:

- **Facet sum**
  This produces a new facet containing all the elements (data and constraints) of the two or more summands, and no additional equations or data. An example of this can be found in section 5.4.1, and the discussion presented in the preceding section illustrates some of the semantic concerns which arise from this particular interaction. In particular, when the summands share common data or constraints, the question of how to capture this shared information must be addressed.

- **Facet inclusion**
  This refers to one facet including another in its declarations, either in the form of a facet variable or simply by referencing it in an equation. In this latter case, the enclosing facet can view all variables declared as exported by the included facet. In accordance with usual scoping rules, those variables declared as private are not visible by any including facet. By contrast, the exported variables can be constrained by the enclosing facet. One of the most interesting areas currently under investigation is the issue of what changes in the included facet are visible to the enclosing facet. This is of great interest when dealing with reusable components. An example of facet inclusion is a Rosetta facet that serves as a two-bit adder:

```plaintext
facet one-bit-adder :: state-based
(x,y,cin,z,cout::bit) is begin
T1: znext(s)=x XOR y XOR cin;
T2: coutnext(s) = x AND y;
end one_bit_adder
```
facet two-bit-adder :: state-based
(x0,x1,y0,y2,z0,z1,c::bit) is
private cx::bit;
begin
L1: one-bit-adder(x0,y0,0,z0,cx);
L2: one-bit-adder(x1,y1,cx,z1,c1);
end two-bit-adder

- Facet variables
  - Facet variables are ordinary variables of type facet, and declared within any other facet. These are predicted by the developers to be of use in localising points of failure in a system.
  - Facet instantiation
  - This refers to the situation where one facet will leave a parameter or other exported variable uninstantiated, and a value for this variable will be given by an external constraint. This external constraint is imposed by another facet which includes the facet in question. Rosetta deems instantiation simply to be the adding of constraints in order to partially or fully determine the value of variables.

One of the major areas of interest is defining the semantics for facet interactions where the facets concerned are from different domains, and so do not share a "common vocabulary". This paper does not cover this issue in detail, but provides the foundations for such work.

2.4 Requirements

There are two central tenets of the semantics of Rosetta, which must be met by whatever framework we choose to provide the formal basis of these semantics.

- Firstly, it is possible for two superficially different specifications to in fact describe precisely the same behaviour. (In the language of component-based specifications, both specifications will be satisfied by precisely the same components or models.) For example, the specifications may use different variable names, or express equivalent axioms slightly differently.

- Secondly, a component or model, which satisfies one specification is said to satisfy all abstractions of this specification. In other words, as long as a component demonstrates the behaviour prescribed in a specification then regardless of what other behaviours the component displays, it is said to satisfy the specification.

The formal semantics must therefore take into account exactly when two Rosetta facets are to be considered to be equivalent, and what the relationships between a facet, such as state-based and its extensions such as increment might be. A full discussion of Rosetta can be found in the official documentation (Alexander et al. 2000), however, it is to be emphasised that this is a language under construction. Lastly, we note that this paper deals with the semantics of the state-based domain only. An alternative theory (Kong, Alexander & Menon 2003) based on algebras (Jacobs & Rutten 1997) has been proposed which attempts to provide a universal semantics, but this is as yet incomplete.

3 Semantic Foundations

3.1 Algebras

Algebras have historically been of great significance when considering the semantics of specification languages, primarily as they allow us to model specifications without recourse to implementation details. Rosetta is no exception, and the representation we choose for Rosetta facets is based on the notion of a many-sorted abstract algebra (Horebeek & Lewis 1989). In essence, we seek to associate a facet with one or more algebras which model the behaviour it describes. We provide a brief description of the formal properties associated with such an algebra A.

A is associated with a signature (S, Δ) where S is a set known as the set of sorts, and Δ is an S x S-indexed family of sets {Δi,j} for i ∈ S, j ∈ S. For f ∈ Δi,j we write f : i → j for the sake of clarity. An algebra A associated with a signature Σ = (S, Δ) is often called a Σ-algebra.

A itself is defined by a denotational function γA which associates each sort in S with a set, and each operator name in Δi,j with a function. Thus, A consists of an S-indexed family of sets AS = {A1,...,An}, where γA(Si) = Ai which are known as the carriers, and a S x S-indexed family ΔA of functions δi,j : i → Aj where i ∈ S.

We can also define a set of variables Xsi for each si ∈ S, which together make up the family of sets of variables of A, namely XA. With this in mind, we can define the term-language TL A of A:

- ∀X ∈ S∀x ∈ Xsi, x ∈ TLA i.e. variables are elements of the term-language.
- ∀f ∈ ΔA, f ∈ TLA i.e. nullary operator names are elements of the term-language.
- ∀f ∈ Δ such that f : S1 x ... Sn → Sl, if t1 ∈ TLA and γ(f(1),...sn) ∈ Al, the function f(t1,...,tn) ∈ TLA i.e. operator names taking other elements of the term-language as parameters are themselves elements of the term-language.

The denotation function γA can be extended such that in addition to the previous functionality, it also associates each variable-free element (term) of the term-language with an element of one of the carrier sets. Given a set of variables for each sort in S, we define an assignment to be a family of mappings indexed by the set S, from the set of variables of a sort to the carrier for that sort:

θ = {θ1, ..., θn} where θi : Xsi → Ai

We can also require that the operators and elements of A1, for all si ∈ S mentioned above obey a family of equations or axioms, TA. These axioms together with the signature form the presentation PA = (SA, ΔA, TA). Each axiom in TA is of the form

t1 = t2, where t1, t2 ∈ TLA

Each axiom, therefore, can be seen to equate two terms of the algebra and is used to define the behaviour of the operators. An algebra is said to satisfy a presentation P = (S, Δ, Υ) if

- the signature of the algebra is (S, Δ) and
- ∀(t1 = t2) ∈ Υ and ∀θ ∈ Υ γ(θ(t1)) = γ(θ(t2)) i.e. for all possible assignments, t1 and t2 are denoted by the same element of the algebra.

The category of algebras which satisfy a presentation P is known as the variety over P, and the category of algebras which are denoted by a given signature Σ is represented by AlgΣ.
Given a signature $\Sigma = \langle S, \Delta \rangle$, a $\Sigma$-homomorphism from $\Sigma$-algebra $A \equiv \langle A_s, A_\Delta \rangle$ to $\Sigma$-algebra $B \equiv \langle B_s, B_\Delta \rangle$ is a family of mappings $\{ f_i : A_i \rightarrow B_i \}$ where $f_i : A_i \rightarrow B_i$ is from the elements of carrier set $A_i$ to the elements of $B_i$ such that
$$\forall \in \Delta_{ij}$, where $i = S_1, \ldots, S_m$, and $\forall \in A_{i_k}$ $f_j(\gamma(\eta_\delta)(f_1(t_1), \ldots, f_m(t_m))) = \eta(\delta)(f_1(t_1), \ldots, f_m(t_m))$.
For a more detailed discussion of algebras, see (Horebeek et al. 1989).

### 3.2 Institutions

Algebras alone do not provide us with the sufficient capabilities to provide a semantic basis for Rosetta. While we can associate various algebras with facets, we need a method of unification, which allows us to see these associations not as isolated mappings, but as part of a projection of Rosetta code into a formal mathematical domain. For this, we turn to institutions.

Informally, an institution is a way of relating a list of elements, a family of constraints on these elements, and a family of models which interpret the elements and so satisfy these constraints in some manner. For example, when utilising algebras, it is useful to examine the the equational logic institution. Here a presentation $(\Sigma, \Gamma)$ (section 3.1) is said to be satisfied by a family of algebras. In this example, these algebras are the models, and satisfy every $\Sigma$-sentence in the presentation.

The utility of institutions is that they provide a formal framework in which to express the idea that truth should be invariant under a change of notation. That is, the actual symbols in $\Sigma$ should not affect which models satisfy a given $\Sigma$-sentence, provided there is a consistent interpretation of whichever symbols are indeed chosen. In other words, if a $\Sigma$-model is said to satisfy a given $\Sigma$-sentence then, should we change the symbols of this $\Sigma$-sentence to produce a $\Sigma'$-sentence, the notion of truth being invariant under change of notation states that there is a $\Sigma'$-model which satisfies this sentence. Traditionally, sentences translate in the same direction as change of notation, while models translate in the opposite direction.

The utility of this for specification languages in general is immediately apparent. When components are built to satisfy a specification, the symbols used in the specification are irrelevant (provided they are interpreted consistently throughout the component!). We thus need a formal means of expressing when two facets might be considered equivalent. With the emphasis of Rosetta firmly on code reuse and extension of existing facets, we are also interested in how any alteration of a specification would affect the set of models which satisfy this specification. Institutions offer a convenient vocabulary to express these ideas and allow us to take steps towards formalizing the different notions of equivalence and satisfaction.

More formally, an institution (Goguen & Burstall 1992) consists of:

- A category $\text{Sign}$, with objects known as signatures.
  
  Each signature $\Sigma$ provides a list of symbols which are the “building blocks” of the models associated with $\Sigma$.

- A functor $\text{Mod} : \text{Sign} \rightarrow \text{Cat}^\text{op}$, associating each signature $\Sigma$ with a category whose objects are known as $\Sigma$-models, and whose morphisms are $\Sigma$-model morphisms.
  
  Models are entities (like algebras, coalgebras or executable programs) comprised of quantities (like variables) which represent the elements of a signature.

- A functor $\text{Arx} : \text{Sign} \rightarrow \text{Set}$, associating each signature $\Sigma$ with a set whose elements are known as $\Sigma$-sentences, or $\Sigma$-axioms.
  
  These are sentences which relate the symbols in $\Sigma$ to each other that, these sentences usually take the form $(t_1, t_2)$, where $t_1, t_2$ are elements of $\Sigma$. A model associated with $\Sigma$ will then implement this by using its definition of equality.

- A satisfaction relation $\models_{\Sigma} \subseteq \text{Mod}(\Sigma) \times \text{Arx}(\Sigma)$ such that
  \[ \forall \Sigma, \Sigma' \text{ in } \text{Sign}, \text{ morphisms } \phi : \Sigma \rightarrow \Sigma', \forall m \in \text{Arx}(\phi)(m') \iff \text{Mod}(\phi)(m) \models \phi(\Sigma')(m' \models e) \text{ for each } e \in \text{Mod}(\Sigma) \text{ and each } e \in \text{Arx}(\Sigma).
  \]

  The satisfaction relation pairs a model with all the sentences which it satisfies, regardless of what symbols are used to express these sentences. For convenience of notation we may occasionally use $\phi(e)$ in place of $\text{Arx}(\phi)(e)$.

A pictorial representation of this appears in figure 2. We see that the satisfaction relation pairs models with sentences and keeps this pairing invariant under signature morphisms. As an example, we show how the equational logic algebras of section 3.1 can be considered as an institution. Here $\text{Sign}$ is the category of algebraic signatures, which has algebraic signatures (Horebeek et al. 1989) as its objects and signature morphisms as morphisms.

There is also a functor $\text{Eqn} : \text{Sign} \rightarrow \text{Set}$ that associates each signature $\Sigma$ with the set of all $\Sigma$-sentences. With a signature $\Sigma$ we can associate a set of algebras (section 3.1), (known as $\Sigma$-algebras and said to be denoted by $\Sigma$). From this, we can form a category $\text{Alg}_{\Sigma}$ with $\Sigma$-algebras as objects, and $\Sigma$-homomorphisms as morphisms.

Let $\text{Cat}$ be the category in which all objects are $\text{Alg}_{\Sigma}$ for some $\Sigma \in \text{Sign}$, and morphisms are these functors $\phi : \text{Alg}_{\Sigma_1} \rightarrow \text{Alg}_{\Sigma_2}$. (Note: all objects in this category are categories themselves.)

We now define a functor $\text{Alg} : \text{Sign} \rightarrow \text{Cat}^\text{op}$, which associates each signature $\Sigma = (S, \Delta)$ with the category $\text{Alg}_{\Sigma}$. $\text{Alg}$ also associates a signature morphism $\phi : (S, \Delta) \rightarrow (S', \Delta')$ with the functor $\text{Alg}(\phi) : \text{Alg}_{\Sigma} \rightarrow \text{Alg}_{\Sigma'}$ that

1. Sends a $\Sigma'$-algebra $\langle A_{S'}, A_{\Delta'} \rangle$ to the $\Sigma$-algebra $\langle A_{S}, A_{\Delta} \rangle$ with $A_{S} = A_{S}(S, \Delta)$ and $A_{\Delta} = A_{\Delta}(S, \Delta)$.
2. Sends a $\Sigma'$ homomorphism $h' : A' \rightarrow B'$ to the $\Sigma$-homomorphism $h \circ \text{Alg}(\phi)(A') \rightarrow \text{Alg}(\phi)(B')$ defined by...
Essentially the above definition states that \( \text{Alg} \) acts on signature morphisms to ensure that a \( \Sigma \)-algebra is associated with a \( \Sigma \)-algebra where all of the carrier sets are also carrier sets for the \( \Sigma' \)-algebra, and that applying a \( \Sigma' \) homomorphism \( k' \) has the same effect (in terms of the resulting carrier sets) as applying \( \text{Alg}(k') \). From these definitions, we have generated the algebraic institution, with \( \text{Sign} \) the category \( \text{Sig} \), \( \text{Mod} \) the functor \( \text{Alg} \), \( \text{Sen} \) the functor \( \text{Eqn} \) and with the usual notion of algebraic satisfaction. That this fulfills all requirements for the Satisfaction Condition was shown by Goguen (Goguen & Burstall 1984).

### 3.3 Theories

When providing a semantic basis for Rosetta, we aim to associate each facet with a *facet signature* consisting of all the declarations visible within the facet, and a set of axioms consisting of those constraints or equations which govern the behaviour of these variables. Institutions allow us to capture the idea that altering the symbols of a specification does not alter the behaviour of the components which satisfy this specification. In addition, we also need some means of associating a signature with a set of equations, in a more unified framework than that offered by algebraic presentations. This is given by the use of *theories*. If \( \Gamma \) is some institution, such that \( \Sigma \) is a signature in the category of signatures associated with \( \Gamma \), then a \( \Sigma \)-theory is a pair \( (\Sigma, E') \) where \( E' \) is a *signature morphism* \( \phi : \Sigma \rightarrow \Sigma' \) such that \( \forall e \in E \), \( \phi(e) \in E' \).

We define a functor \( \text{Sign} : \text{Th} \rightarrow \text{Sign} \) with the following action:
- \( \text{Sign}(\Sigma, E) = \Sigma' \)
- \( \text{Sign} \) sends \( \phi \) as a theory morphism to \( \phi \) as a signature morphism

#### 3.3.1 Colimits

Colimits are used in many areas of category theory (Fokkinga 1992) and are an example of a categorical construction by *initiality*. Given a number of objects which all have a certain property \( p \), the initial object \( A \) is the object for which there is *precisely* one morphism from \( A \) to *every* other object with property \( p \). Of course, there may be no such object \( A \), but in the case where there is such an object, it is determined uniquely up to isomorphism. Colimits are perhaps the most common such categorical construction.

To define a colimit we must first define *diagrams*. A diagram in a category \( C \) is a directed graph \( G = \{\text{Nodes}, \text{Edges}\} \) and a labelling function \( L_C : G \rightarrow C \) which labels nodes with objects from \( C \) and edges with morphisms from \( C \) such that

\[
\forall d_1, d_2 \in \text{Nodes} \text{ such that } A_1 = L_C(d_1), A_2 = L_C(d_2) \text{ and } \alpha = (d_1, d_2) \in \text{Edges} \text{ such that } \alpha : A_1 \rightarrow A_2.
\]

For simplicity’s sake, we consider a diagram \( D \) in \( C \) to be the function \( L_C \) given above, treated as a functor (a map from one category to another).

Given a category \( C \), a graph \( G \) and a diagram \( D : G \rightarrow C \) we define a category \( \text{Vd} \) where:

- Objects in \( \text{Vd} \) are *cones* where a cone \( \Pi \) consists of some object \( A \in C \) known as the apex of \( \Pi \) and a family of morphisms in \( C \): \( \Gamma : \Pi : D(d) \rightarrow A \), one for each \( d \in G \) such that \( \forall e \in \{d_1, d_2\} \in G, D(e) \circ \Gamma_2 = \Gamma_1 \)

* i.e. such that for each edge \( e = (d_1, d_2) \), figure 3 (a) commutes in \( C \)

* A morphism \( f : \Pi \rightarrow \Psi \) in \( \text{Vd} \) is a morphism \( \beta : A \rightarrow B \) in \( C \) where \( A \) and \( B \) are the apexes of \( \Pi \) and \( \Psi \) respectively, such that for each \( d \in G \), \( \Pi \circ f = \Psi \circ \gamma \) i.e. such that figure 3 (b) commutes in \( C \)

A co-limit is then an initial object in \( \text{Vd} \), and the apex of this object is a *colimit object*. We say that a category \( C \) is (finitely) *co-complete* if for each (finite) category \( \text{Vd} \) built upon \( C \), \( \text{Vd} \) has an initial object (in other words, iff each \( C \) has colimits of each (finite) diagram).

#### 3.3.2 Theories and colimits

We propose to use \( \text{Th} \), the category of theories over the equational algebra institution, extensively in this paper. Many of the results will depend on the fact that \( \text{Th} \) has colimits and because of the fundamental importance of this result, we present a proof. This is a variation on an earlier proof (Goguen & Burstall 1980).

To show that a category \( C \) has finite colimits, it is sufficient to show that it has coequalisers (Fokkinga 1992), an initial object, and binary coproducts (MacLane 1971). This is used in the following proof, showing that the category of theories has finite colimits. We also show how to construct a theory that is the coproduct of two given theories.

**Proof**

- Since \( \text{Sig} \) (the set of algebraic signatures) is co-complete (Goguen and Burstall 1991), \( \exists \Sigma_\emptyset \), the initial object in \( \text{Sig} \). The initial object of \( \text{Th} \) is then \( (\Sigma_\emptyset, \emptyset) \) where \( \emptyset \) is the empty set, as this is the unique set contained in every other.

- Given two theories \( (\Sigma_1, E_1) \) and \( (\Sigma_2, E_2) \), we find the coproduct as follows: Let \( \Sigma_3 \) be the coproduct in \( \text{Sig} \) of \( \Sigma_1, \Sigma_2 \) by means of the morphisms \( \delta_1 : \Sigma_1 \rightarrow \Sigma_3 \) and \( \delta_2 : \Sigma_2 \rightarrow \Sigma_3 \). This is known to exist, as \( \text{Sig} \) is finitely co-complete, and a coproduct is simply a colimit over a certain diagram. Then, the co-product of two theories \( (\Sigma_1, E_1), (\Sigma_2, E_2) \) can be seen to be \( (\Sigma_3, (\delta_1 E_1) \cup \delta_2 E_2)) \)}
\(\delta_1(E_3)\). Note that, as we are using theories rather than presentations, it is not necessary to apply the Presentation Lemma (Goguen and Burstall 1980) in this proof.

- Given two theory morphisms \(\delta_1, \delta_2 : (\Sigma, \delta) \rightarrow (\Sigma', \delta')\), the coequaliser in \(\text{Sig}\) of \(\delta_1, \delta_2 : \Sigma \rightarrow \Sigma'\) is \(\delta : \Sigma \twoheadrightarrow \Sigma'\). It is clear then that the coequaliser in \(\text{Th}\) is the theory morphism \(\delta : (\Sigma, \delta) \rightarrow (\Sigma', \delta')\), by an examination of the requirements for a theory morphism and initially.

From this, we see that \(\text{Th}\) is finitely co-complete. Moreover, this is due solely to the properties of \(\text{Sig}\), and so holds for any co-complete category of signatures, not just the category of equational logic algebraic signatures.

4 Basic Semantics of Rosetta Facets

4.1 Semantic Requirements

We use the previous definitions as a framework to model Rosetta facets and their interactions. In particular, we associate each Rosetta facet with a theory consisting of a signature and collection of axioms. As Rosetta is a language which relies on first-order logic to express the behaviour of any variable, it makes sense to look to the equational logic institution to see what we can glean that may be of use.

In section 2.4 we presented the informal conditions which must be met by the semantic basis of Rosetta. It is apparent that, barring the formal proofs, the first of these conditions is met by institutions, which provide a notion of truth invariant under change of notation. In other words, two superficially different specifications, or sets of sentences, are satisfied by the same models if all that has occurred is a change in notation. The second condition is satisfied by the use of theories to model specifications. If there is a theory morphism \(\gamma : Th_A \rightarrow Th_B\), then the set of models which satisfies \(A\) is contained within the set of models which satisfies \(B\). (There is a Galois connection (Goguen and Burstall 1992) between the relation \(\alpha\) which given a collection \(E\) of \(\Sigma\)-sentences, produces a set of models \(M\) which satisfy every \(e \in E\), and the relation \(\beta\) which, given a collection of \(\Sigma\)-models \(M\), produces the set of \(\Sigma\)-sentences which are satisfied by every model in \(M\).)

With this in mind, we propose to adapt the equational logic institution to provide a formal semantic basis for Rosetta.

4.2 Facet Signatures

It is possible to associate any legal Rosetta facet \(f\) with an algebraic signature \(\Sigma_f = \langle S_f, \Delta_f \rangle\), \(S_f\), the set of sorts is made up of the labels by which \(f\) refers to the following elements of Rosetta which are visible within the facet. This includes declarations which are placed in a domain the facet extends:

- Elemental types such as \(\text{Int}, \text{Char}, \text{Bit}, \text{Real}, \text{Function} \ldots\) (Alexander et al. 2000)
- Values of these sorts. With the strict typing of Rosetta as a language, we treat them as unique sorts. This fits with the strict typing of Rosetta as a language.
- Sets and sequences of elemental types
- \(\bot\), or \(\text{ERROR}\)
- Constructed types (such as \(\text{Tree}\))
- \(\text{State}\)

- \(\text{Function}_{\lambda,s} \) for \(\lambda \in S_f^*, s \in S_f\)

\(\text{State}\) refers to the state of a (state-based) facet and is used to dictate how the values of certain variables change relative to the current values. This is discussed further in (Alexander et al. 2000) and (Ashenden et al. 2003). For convenience sake, we partition \text{Function} into sets of functions according to their rank and arity (i.e. \(\text{Function}_{\lambda,s} \subseteq \text{Function}\))

\(\Delta_f\) consists of a set of operators which denote the different variables and functions declared in \(f\). Thus, for each Rosetta variable \(x\) of type \(T\) is denoted by a function \(get_x \in \Delta_f\) where \(get_x : \text{State} \rightarrow T\). This, of course, applies to Rosetta function variables as well. Moreover, we define some additional operators which are the “traditional” operators of algebraic theory. These take the form \(eval_{\lambda,s} : \text{Function}_{\lambda,s} \rightarrow s\).

The purpose of these operators (or more precisely, the functions an algebra associates with them) is to describe the behaviour of a given function with rank \(\lambda\) and arity \(s\).

For any given facet \(f\) with signature \(\Sigma_f\) we can also obtain a presentation \(E_f\) from all axioms which constrain the value of variables within it. This includes code actually written in the body of the facet itself as well as code from a domain the facet extends and axioms due to instantiation. This presentation is to be equationally closed. That is, if it is possible for an equation \(t_1 = t_2\) where \(t_1, t_2\) are variables within \(f\) to be derived by equational logic from the code, then this equation forms part of \(E_f\).

Thus, we have associated a facet \(f\) with \(\langle \Sigma_f, E_f \rangle\).

4.3 Rosetta Institutions

Building on the generation of an algebraic signature, we demonstrate below how we associate a Rosetta facet with an institution.

- Let \(\text{Sign}\) represent the category of all Rosetta signatures, generated as described in section 4.2. Morphisms within this category are, of course, signature morphisms.
- The functor \(\text{Alg} : \text{Sign} \rightarrow \text{Cat}^\text{P}\) associates each signature \(\Sigma\) with the variety of algebras which are denoted by \(\Sigma\). The action of \(\text{Alg}\) is defined in section 3.2.
- The functor \(\text{Sen} : \text{Sign} \rightarrow \text{Set}\) associates each signature \(\Sigma\) with the set of \(\Sigma\)-sentences which may be legally expressed in Rosetta.

We define the satisfaction relation for this institution to be the equational logic satisfaction relation, linking \(\Sigma\)-models and \(\Sigma\)-sentences. As this institution, \(R\), is built so closely upon the equational logic institution, we refer the readers to (Goguen and Burstall 1992) for further discussion.

4.4 Rosetta Theories

Earlier, it was shown that each facet \(f\) can be denoted by \(\langle \Sigma_f, E_f \rangle\), where \(\Sigma \in \text{Sign}\) and \(E_f\) is closed, it follows by definition that each facet can be denoted by a theory over institution \(R\). By associating a facet with an equationally closed set of axioms, we define that facets are unique up to axiom abstraction (Lawvere 1963). However, we do not apply signature abstraction. The result of these choices is that if two facets generate the same signature (so they contain exactly the same names of variables, constants, functions and datatypes) and the axioms (constraints) of
either facet can be deduced by equational reasoning from the axioms (constraints) of the other, then these two facets will be associated with the same theory. On the other hand, if two facets generate different signatures then, irrespective of their axioms, they will be associated with different theories. Similarly, two facets which generate the same signature but do not generate equivalent sets of axioms (so do not contain equivalent constraints) will be associated with different theories.

5 Semantics of Combining Rosetta Facets

5.1 Relations between individual facets

As we have associated each Rosetta facet with a theory, it makes sense to base the semantics upon Th, the category of theories. A Rosetta facet corresponds to a theory, and Rosetta extensions (Alexander et al. 2000) correspond to certain theory morphisms. In general, a theory morphism can be said to be a relationship between two theories where the equations which hold in the source theory also hold in the target theory, up to renaming (change of notation). In other words, a theory morphism is a signature morphism with the additional constraint that it can only add information to the source theory. Rosetta extensions, on the other hand, are required to conform more to the intuitive notion that an extension should consist only of addition of elements and information. In other words, should Rosetta facet f1 extend Rosetta facet f2, we not only require that f1 contain at least all the information of f2 but also at least all the variables of f2. This is represented by those theory morphisms which are injective. The morphisms which are not injective have no parallel in Rosetta. In this way we define a category RTh which is a subcategory of Th. The objects of RTh are theories and the morphisms are those theory morphism which are injective when considered as signature morphisms.

To emphasise the implications of using this category RTh, we discuss how some elementary properties of Rosetta facets are represented in the category. Due to its origin as a specification language, Rosetta like VHDL places an emphasis on what we call behavioural equivalence. Informally, we say that two facets are behaviourally equivalent if they have the same number and type of variables, and after applying some renaming function, the closures of the sets of axioms of both imply each other. As a formal definition, let two facets f1, f2 be denoted by theories (Σ, E), (Σ', E'). These are behaviourally equivalent if there is a theory morphism α : Σ → Σ' where

- α is bijective as a signature morphism
- α(E) ↔ E'

From this, it is easy to form the alternative definition: Two theories correspond to behaviourally equivalent facets iff they are isomorphic when considered as objects of Th.

This means that, should it become necessary to consider behaviourally equivalent facets as semantically identical, this can be achieved simply by taking a transversal of the isomorphism classes.

5.2 Facet extensions

Rosetta encourages the position that extension of a facet is not just a matter of adding information but should instead be seen as further constraining a specification. This originates from the idea that as long as a component does not violate any of the constraints of a specification, it is said to meet this specification. Thus, by extending a specification, we can be said to be imposing further constraints and, in practice, to be reducing the size of the set of components which will satisfy it. Taken to the extreme, it can be said that any behaviour whatsoever satisfies the null (empty) specification. We can formalize this by referring back to the institution framework.

Much work on algebras has concentrated on initial algebras, which are those with the "no junk" and "no confusion" properties. These state respectively that there must be no element of the algebra (i.e. no function or element of any carrier set) which is NOT the denotation of some element of the term language, and (no confusion) that two elements of the term language are denoted by the same element of the algebra if there is an axiom to this effect in the presentation.

Thus, when referring to a presentation (Σ, F), the initial algebra I which satisfies this presentation is the algebra which satisfies only this and nothing more. While initial algebras certainly are considered interesting properties, such as uniqueness up to isomorphism, this definition of satisfaction is perhaps too strict for the satisfaction of language community. As described earlier, the category \[ Mod(\text{Sig}) \] whose elements we have chosen to serve as models for Rosetta facet signatures is the category of all algebras which are denoted by the signature in question. Moreover, the satisfaction relation pairs up a model m and individual \( \Sigma \)-sentences without considering which other \( \Sigma \)-sentences \( m \) may also satisfy. For a given theory (\( \Sigma, E' \)), we see that the models which the satisfaction relation pairs with \( e \) for all \( e \in E' \) are those algebras which satisfy at least all \( e \in E' \) (in other words, the variety over the presentation).

This is the precise condition for satisfaction required for specification languages such as Rosetta. It is clear that adding axioms or signature elements to a theory (\( \Sigma, E \)) to obtain (\( \Sigma', E' \)) means that the set of models \( M \) which satisfies (\( \Sigma', E' \)) is contained within the set which satisfies (\( \Sigma, E \)). When we also consider renaming, this results in the following lemma.

**Lemma 1**

Let \( \alpha \in RTh \) where \( \alpha : (\Sigma, E) \rightarrow (\Sigma', E') \), and let

- \( A = \{ x \in M od(\Sigma) \mid \text{s.t. } x = e, \forall e \in E \} \)
- \( B = \{ x \in M od(\Sigma') \mid \text{s.t. } x = e, \forall e \in E' \} \)

Then \( \alpha(B) \subseteq A \).

One area where this is clearly demonstrated is when the constraints placed upon a variable do not uniquely identify the value that the variable takes. Any model which assigns a value to that variable which does not violate any of the constraints is said to satisfy the specification. Thus, although each model assigns only one value to that variable, all possible legal values are represented in the set of models which satisfy the specification. This method of dealing with nondeterminism avoids many of the pitfalls associated with stricter methods (Goguen and Burstall 1992).

5.3 Facet interactions

While the previous discussion has centred around individual facets and how they are represented using these semantics, the structure that such a basis gives to interactions between facets is a primary benefit. There are several different types of facet interactions, including facet sum, facet implication, facet inclusion and facet variables. Interactions are relevant when a
facet represents a component in a system which is required to share information or constraints with other components. Of these, facet implication is probably the simplest. A Rosetta facet $f_1$ is said to imply another facet $f_2$ if any axiom which $f_1$ includes is also included by $f_2$. Contrary to intuition, this does not mean that any allowable behaviour of $f_1$ is also allowable behaviour of $f_2$, but rather its converse. This is one situation where the terminology of Rosetta diverges from the usual terminology of specification languages. However, it is very simply expressed in terms of $\text{RTh}$: Let $f_1$ and $f_2$ be denoted respectively by theories $(\Sigma_1, E_1)$, $(\Sigma_2, E_2)$. Then $f_1$ implies $f_2$ iff there exists a morphism $\alpha \in \text{RTh}$ such that $\alpha : (\Sigma_1, E_1) \to (\Sigma_2, E_2)$.

Another useful facet interaction is facet sum. Given two facets $f_1$ and $f_2$ then we define a diagram $D : G \to \text{Th}$ where $D$ determines a subcategory of $\text{Th}$ consisting of a user-defined subset of the extension hierarchies of each of $f_1$ and $f_2$. We now define $f_1 + f_2$ to be the facet that is denoted by the colimit object in $\text{Th}$ of $\forall D$. Since the colimit object is defined up to isomorphism, we see in fact what we have obtained as the colimit of $f_1 + f_2$ is an isomorphism class within $\text{Th}$. In other words, it is determined uniquely up to signature abstraction and axiom abstraction. This is behaviourally equivalent in its purest form and is a great advantage, since this particular construct of component summation is only used when one is interested in the observable behaviour of the resulting component. Due to the Galois connection mentioned in section 4.1, all isomorphic theories over a given institution are associated with the same set of models, so from a specification point of view, it does not matter that we don’t uniquely identify $f_1 + f_2$.

5.4 Examples

We provide some examples to show exactly what the resulting theory $(\Sigma, F)$ would look like (for simplicity’s sake we choose one colimit object out of the many possible isomorphic such). 5.4.1 Example 1

Suppose we wish to sum Rosetta facets $f_1$ and $f_2$ where these have no common inheritance (that is, there is no facet $f_3$ such that $\alpha_3, \beta_3 \in \text{Th}$ such that $\alpha : (\Sigma_3, E_3) \to (\Sigma_1, E_1)$ and $\beta : (\Sigma_3, E_3) \to (\Sigma_2, E_2)$). Then the coproduct $(\Sigma, F)$ is the disjoint union of $(\Sigma_1, E_1)$ and $(\Sigma_2, E_2)$:

$f_1 + f_2$ is:

```plaintext
bool:: type
false, true:: bool
+
bool * bool1 -> bool
begin
true+false = false
end
```

$f_2$ is:

```plaintext
nat:: type
0, 1:: nat;
inc:nat -> nat;
begin
inc(0) = 1
end
```

The sum is:

```plaintext
f3:: bool1, nat:: type
false, true:: bool
```

5.4.2 Example 2

Suppose we wish to sum Rosetta facets $f_1$ and $f_2$ where these share some common data types or variables (for example, both extend facet $f_1$-bool above, which declares the Boolean datatype), albeit with possible renamings. Due to the commutativity requirements depicted in figure 4, we see that there can be only one copy of this shared datatype and its operations in the colimit object. This can be more generally expressed as:

If Rosetta facets $f_1$ and $f_2$ inherit from a common facet $f$, then $f_1 + f_2$ will contain only one copy of any elements which originate in $f$. Let $\delta_1 : f_1$-bool $\to f_1$, $\delta_2 : f_1$-bool $\to f_2$ as depicted below:

```plaintext
f1 is:
bool1:: type
false1, true1:: bool1
+
bool1 * bool1 -> bool1
x:: bool1
begin
true1+false1 = false1
end
```

```plaintext
f2 is:
nat:: type
bool2:: type
false2, true2:: bool2
+
bool2 * bool2 -> bool2
y:: bool2
begin
true2+false2 = false2
end
```

The sum of $f_1$ and $f_2$ is:

```plaintext
f1+f2 is:
f1:: bool1, nat:: type
false, true:: bool1
+
bool1 * bool1 -> bool1
begin
true+false = false
end
```

For ease of notation we let $f_1$ represent the object in Th which represents facet $f_1$. Now, $f_1 + f_2$ is a colimit object in diagram 4, where the actions of $\delta_1$ and $\delta_2$ are as defined above. $f_1 + f_2$ contains only one copy of $f_1$-bool, as well as the variable $z$ which is mapped to by $x$ and $y$, and the type $\text{nat}$, defined in $f_2$. This approach parallels the data universe

![Figure 4: Colimits of theories which share elements](image-url)
proposed in [Goguen and Burstall, 1992] in that we want interacting facets to share the same notion of datatypes. This gives them a common language with which to communicate and enables information to be shared.

5.5 Extensions revisited

In general when we extend two facets $f_1$, $f_2$ to produce a third $f_3$ (as in facet sum), one of the most pressing questions is that of interference, or overlap of variables. This refers to the situation where an element $x$ of $f_1$ maps to the same element $z$ of $f_1+f_2$ as does some element $y$ of $f_2$. This, naturally, means that any constraints which $f_1$ applied to $x$ and any constraints which $f_2$ applied to $y$ must all apply to $z$. Depending on the properties we want the resulting facet to display, we can allow as much or as little overlap as required, and can also add additional constraints to obtain more properties. One common scenario is designing a system where one component, such as a switch, is inherited by many others which also define additional constraints and properties of the switch. We may wish to add all these facets together to produce a facet which has one highly-constrained switch, or alternatively to produce a facet which has many copies of differently constrained switches. In essence this can be thought of as the difference between producing a facet which has the maximum number of variables all minimally constrained, and producing that facet which has the minimum number of variables all maximally constrained. By defining different diagrams $D$ in different situations, the user is able to choose which particular facet is denoted as being the apex of the initial object in $VD$, and is thus represented by $f_1 + f_2$.

6 Further Work

The previous discussion has focussed primarily on individual facets and extremely simple interactions. However, the advantage of Rosetta over other requirements and simulation languages is the depth and number of interactions made possible. We are now working to extend the above framework to facet inclusion and other forms of facet interaction.

6.1 Facet Inclusion

Facet inclusion incorporates instantiation, or the adding of axioms to other objects. Specifically, we see that it is possible for the including facet to constrain the value of variables which are declared in the included facet. Consider the following example:

```
facet f1 is
x:: int;
begin
T1: f2-template;
T2:T1.y=x;
end f1
```

We observe that the approach outlined in this paper does not consider that there may exist axioms which equate a variable $x$ in one facet to a variable $y$ in another, where $x$ and $y$ are not both in the signature $\Sigma$ of one of these facets. Constraints of this form cannot be expressed as $\Sigma$-sentences, yet appear often in the case of facet inclusion. However, as we can see above, an instantiated facet in Rosetta never exists in isolation. It therefore makes little sense to consider the theory of $T_1$ above without associating it with the theory of the including facet. (It is worth noting at this point that the difficulty arises only when associating a theory with the included facet. The signature of the including facet $f_1$ consists of all those datatypes and variables observable to it which includes those declared and exported from facet $f_2$-template).

As a result, we do not separate the above declaration into objects which correspond to the included and including facets, but simply associate only one theory $T = (\Sigma, E)$ with the entire of the declaration. $\Sigma$ contains all sorts and functions declared in either $f_1$ or $f_2$-template, while $E$ contains all axioms which are part of either $f_2$-template or $f_1$. To formally express this, we must consider the impact of hidden variables and sorts (those which are not exported (Alexander et al., 2000)), which is discussed below.

6.2 Hidden Data

One of the major areas of interest is in determining replacement components that is, given a facet $f_1$ which includes another facet $f_2$, we are interested in determining which other facets $f_1$ observes as being equivalent to $f_2$ under inclusion. This form of behavioural equivalence takes into account which elements of a signature are visible and which are hidden, and in this respect is closer to the notion of bisimulation (Kurz, 1998). We propose as future work some analysis of this using the framework of this paper, and comparing to the coalgebraic approach presented in (Kong, Alexander, Mencin, 2005). It also highlights the importance of data hiding and the notions of a hidden signature (Goguen & Malcolm, 2000).

There are some elements which the including facet $f_1$ will never be able to observe directly, such as the state of the included facet $f_2$. Any change in these it can only observe indirectly, if a change is produced in a variable visible to $f_1$. $f_1$ will also not be able to observe datatypes of the form of constructed types declared within $f_2$ (and hence no variables of these types) or variables if either of these are not exported. Given this, we define the hidden signature of $f_2$. $\Sigma = (\tilde{S} \cup S, \Delta \cup \Delta)$, where $S$ and $\Delta$ refer respectively to those sorts and functions which correspond to datatypes and variables which are NOT exported from Rosetta facet $f_2$. $\tilde{S}$ and $\Delta$ refer to those sorts and functions which are exported. We can expand on this to produce a hidden theory $\langle \Sigma, E \cup F \rangle$, where $E$ is a closed set of equations which relate terms which do not incorporate any hidden elements. We call the visible part of this theory the interface of $f_2$. Note that this interface $(\tilde{S}, \Delta, E')$ itself is also a theory.

A facet $f_2$ is said to be visibly equivalent to $f_2$ if they provide exactly the same interfaces. In a practical sense, this means that $f_1$ can replace the included facet $f_2$ with $f_3$ and there will be no observable difference, from the viewpoint of facet $f_1$. This parallels the notion of views (Goguen & Tracz, 2000), in that if two facets provide the same interface then, regardless of what hidden sorts or functions they may each contain, the visible consequences of any equation satisfied by either theory must be the same for both facets. Similarly, we say $f_2$ visibly satisfies $f_3$ if the interface of $f_3$ contains the interface of $f_2$. In practical terms, replacing an included facet $f_2$ with $f_3$ in this case
would correspond to replacing a simple component with a more complicated one. In this situation, the more complicated component should still perform at least the same functions as the simpler component.

### 6.3 Hidden theories

Looking once again at the example of facet inclusion given above, we see that in order to share information and constraints, the facets in question must share certain sorts, just as they were required to do in the definition of facet sum. We define $\Sigma$ to be the signature of the interface of $f_2$-template, and $\Sigma_1$ to be all signature elements visible within the facet of this example that are also declared in some source other than $f_2$-template (although they may also be declared in $f_2$-template, for example the integers may be declared in a facet that is extended by other facets which help make up the example). Then the signature of $f_1$ is the colimit in $\text{Sig}$ of $\Sigma$ and $\Sigma_1$. Similarly, the theory of $f_1$ is also a colimit in $\text{Th}$.

By altering the definition of a theory to provide the machinery to express hidden sorts and functions, we obtain hidden theories (Goguen and Burstall 1992). A hidden theory morphism is then a theory morphism which preserves those sorts and functions as hidden, which allows us to identify hidden theory morphisms which are visibly equivalent from the observations made by any given facet.

### 7 Conclusion

This paper has proposed a category theory framework for the semantics of the new specification and simulation language Rosetta. We have shown that the appropriate semantics for facets is that of algebraic theories. Further, by considering theories over institutions, we can accommodate the notion of behavioural equivalence, i.e. where two facets describe the same behaviour but with different notation. There is also interest in observational equivalence, and we anticipate being able to support this in the above framework with hidden theories.

Rosetta is distinctive in supporting a rich variety of facet interactions. We have shown how facet extension corresponds to extension morphisms in the category of theories, and how facet sum corresponds to colimits in the same category. Other forms of facet interaction are the subject of further work, both in terms of a more precise definition of the language and in terms of a categorical formulation.

### 8 Bibliographical References

#### References


