# Logical Semantics for CafeOBJ

Răzvan Diaconescu[†] and Kokichi Futatsugi

IS-RR-96-0024S

[†]On leave from the Institute of Mathematics of the Romanian Academy.

# Logical Semantics for CafeOBJ

Răzvan Diaconescu* and Kokichi Futatsugi

Graduate School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Ishikawa 923-12, JAPAN

## 1 Introduction

This document presents the semantics of CafeOBJ system and language (see [11]). CafeOBJ can be seen as a succesor of the famous algebraic specification and programming language OBJ [24, 10] but adding several new paradigms to the traditional OBJ language, such as specification of concurrent systems, object-orientation, and behavioural specification.

CafeOBJ is a declarative language in the same way other OBJ-family languages (OBJ, Eqlog [19, 4], FOOPS [21], Maude [27]) are. Therefore the formal semantics of CafeOBJ follows the same general principle:

> **(P1)** there is an underlying logic[1] in which all basic constructs/features of the language can be rigorously explained.

This intimate relationship between the language and its underlying logic was called "logical programming" by Goguen and Meseguer in [20].

In providing the semantics for CafeOBJ we distinguish between four levels of the language:[2]

- programming "in the small",
- programming "in the large",
- programming "in the huge", and
- environment.

Programming "in the small" refers to collections of program statements (as obtained by *flattening* the individual modules), programming "in the large" to the module interconnection system, programming "in the huge" to the software system composition, and the environment to the set of tools (including methodologies) supporting the process of programming and specification building in CafeOBJ. While programming in the small and in the large require formal mathematical semantics, programming in the huge can be approached semantically using some general techniques developed for programming in the large (see [12]), the environment cannot (and should not) be given formal semantics. However we devote a brief section to the latter because this is an essential aspect of the CafeOBJ system which should be understood in relationship to the former levels. So we formulate the second principle of our semantics:

---

*On leave from the Institute of Mathematics of the Romanian Academy.

[1] Here "logic" should be understood in the modern relativistic sense of "institution" which provides a mathematical definition for a logic (see [14]) rather than in the traditional sense.

[2] This hierarchy was first suggested by Professor Goguen, and we find it very meaningful for structuring our approach to the semantics of CafeOBJ.

**(P2)** provide an integrated, cohesive, and unitary approach to the semantics of programming/specification in the small and in the large.

The third principle refers to the methodology of developing the logical semantics:

**(P3)** develop all ingredients (concepts, results, etc.) at the highest appropriate level of abstraction.

In order to achieve this we make extensive use of the powerful modern semantic tools made available by research in algebraic specification over the past decade, such as institutions and category theory. Institutions make it perfect for developing the semantics of sophisticated systems implementing a multitude of mutually interacting paradigms in a simple, clean, and compact manner. Modern systems, including CafeOBJ, cannot escape a certain degree of complexity and sophistication, however institutions (and more generally, categorical methods) greatly help in retaining a basic simplicity at least at the level of semantics. Moreover, our abstract logical approach permits future extensions of CafeOBJ with other paradigms provided they are rigorously based on logic and they interact well with the existing paradigms; such extensions will still lie within the present semantics.

Finally, this document does not address the detalied mathematical aspects of this semantics (which sometimes could be rather sophisticated) but rather give pointers to other documents backing our claims. However, we provide in the appendices very brief surveys of several key structures; we hope this will make this document more self contained.

### Acknowledgment

We thank Joseph Goguen for several helpful suggestions improving this document. The first author is also grateful to Professor Goguen for the teachings over the past years which made this work possible.

## 2 Main Features of CafeOBJ

This section gives a brief overview of the main features of CafeOBJ, all of them reflecting in the logical semantics. These should be understood in their combination rather than as separated features. Combining some of these features (sometimes all of them!) results in new specification/programming paradigms that are often more powerful than the simple sum of the paradigms corresponding to the individual features. One example is given by [8].

### Equational specification and programming

This is inherited from OBJ [24, 10] and constitute the basis of the language, the other features being built on top of it. As with OBJ, CafeOBJ is *executable*, which gives an elegant declarative way of functional programming, often refered as *algebraic programming*.

### Concurrent systems specification

This is based on Meseguer's *rewriting logic* [27] (abbreviatted **RWL**) specification framework for concurrent system which gives a non-trivial extension of traditional algebraic specification towards concurrency. This feature brings the Maude language [27] close to a subset of CafeOBJ. RWL incorporates many different models of concurrency in a natural, simple, and elegant way, thus giving CafeOBJ a wide range of applications.

### Behavioural specification

Behavioural specification [16, 18] provides another novel generalisation of traditional algebraic specification but in a different direction. Behavioural specification characterise how objects (and systems) *behave*, not how they are implemented. This is achieved by using specification with hidden sorts and a behavioural concept of satisfaction based on the idea of indistiguishability of states that are observationally the same.

### Object orientation

In CafeOBJ there are two sources of object-orientation. The first is given by the rewriting logic á la Maude treatment of objects which is implemantation oriented, the second one is given by the behavioural specification of objects which is more faithful to the principle of state encapsulation.

### Powerful module system

The principles of the CafeOBJ module system are inherited from OBJ which builds on ideas first implemented by the language Clear [2, 3]. CafeOBJ has several kinds of imports, parameterised programming (also allowing integration of CafeOBJ specifications with executable code in a lower level language), views, and module expressions.

### Powerful type system

The type system that allows subtypes based on *order sorted algebra* [22, 15] (abbreviatted **OSA**). The method of *retracts*, a mathematically rigorous form of runtime type checking and error handling, gives CafeOBJ a syntactic flexibility comparable to that of untyped languages, while preserving all the advantages of strong typing. The order sortedness of CafeOBJ not only greatly increases expressivity, but it also provides a rigorous framework for multiple data representations and automatic coercions among them [15].

## 3 The Underlying Logic

Each of the main paradigms implemented in CafeOBJ is rigorously based on some underlying logic; the paradigms resulting from various combinations are based on the combination of logics. This is consistent with the way the semantics of other multi-paradigm declarative languages has been treated, i.e., by combining the underlying logics. The following table shows the correspondence between specification/programming paradigms and logics as they appear in the actual version of CafeOBJ, also pointing to some basic references.

| ABBREVIATION | LOGIC | SPEC/PGM PARADIGM | BASIC REF. |
|---|---|---|---|
| MSA | many sorted algebra | algebraic specification | [13] |
| OSA | order sorted algebra | algebraic specification with subtypes | [13, 22, 15] |
| HSA | hidden sorted algebra | behavioural specification | [16] |
| HOSA | hidden order sorted algebra | behavioural specification with subtypes | [16, 1] |
| RWL | rewriting logic | concurrent algebraic specification | [27] |
| OSRWL | order sorted rewriting logic | concurrent algebraic specification with subtypes | |
| HSRWL | hidden sorted rewriting logic | behavioural concurrent algebraic specification | [8] |
| HOSRWL | hidden order sorted rewriting logic | behavioural concurrent algebraic specification with subtypes | |

There are some enrichment/embedding relations between these logics, which correspond to institution embeddings (i.e., a strong form of institution morphisms of [14, 9]; see Appendix A), and which are shown by the following **CafeOBJ cube** (the orientation of arrows correspond to moving from "less complex" to "more complex" logics).

$$\begin{tikzcd}
& \mathrm{OSA} \arrow[rr] \arrow[dd] & & \mathrm{OSRWL} \arrow[dd] \\
\mathrm{MSA} \arrow[ur] \arrow[rr] \arrow[dd] & & \mathrm{RWL} \arrow[ur] \arrow[dd] & \\
& \mathrm{HOSA} \arrow[rr] & & \mathrm{HOSRWL} \\
\mathrm{HSA} \arrow[ur] \arrow[rr] & & \mathrm{HSRWL} \arrow[ur] &
\end{tikzcd}$$

More rigorously, when dealing with pre-defined data types (which is required for any system having reasonable library support), the semantics must involve *constraint logics* [4, 6]. But since this issue is somehow secondary to our approach, and also because constraint logics can be easily internalised to any of the institutions constituting the CafeOBJ cube (see [6]), we feel that for the purpose of the presentation it is not necessary to add another dimension to the CafeOBJ cube.

HOSRWL embedds all other institutions (and therefore all main features of the language), hence it can be regarded as the institution underlying CafeOBJ; we devote Appendix B to the brief presentation of HOSRWL. However, it is important to consider the CafeOBJ cube in its entirety rather than HOSRWL alone. In a sense, HOSRWL represents the flattening of the cube, and some subtle information on the relationship between the component features is lost in this flattening.[3]

[3]One simple example is given by imports of MSA modules by RWL modules, their denotations should map RWL models to algebras by getting rid off the transitions. This process directly uses the embedding of MSA into RWL and cannot be explained within HOSRWL alone.

For any two vertices of the CafeOBJ cube there is at most one institution embedding in the cube, so the embedding relation between the CafeOBJ cube institutions is a partial order, which we denote by $\sqsubseteq$. Least upper bounds and greatest lower bounds in the CafeOBJ cube will be denoted by $\sqcup$ and $\sqcap$ respectively.

# 4 Programming in the Small

At this level, semantics of CafeOBJ is concerned with the semantics of collections of program statements as given by flattening the individual modules, i.e., discarding any module composition structure. In CafeOBJ we can have several kinds of modules, the basic kinds corresponding to the specification/programming paradigms shown in the table of Section 3 (but discarding the type component):

- equational specification modules,
- rewriting modules,
- behaviour modules, and
- behaviour rewriting modules

The membership of a module to a certain class is determined by the CafeOBJ convention that each module should be regarded as implementing the simplest possible combination of paradigms resulting form its syntactic content (see [26]). This contrasts with Maude's approach which uses keywords for specifying the class of each module.

The table of Section 3 also shows the underlying logic corresponding to each class of modules. Modules can be regarded as finite sets of sentences in the underlying logic. This observation enables us to formulate the principle of semantics of CafeOBJ programming in the small:

> **(S)** We identify between modules and theories generated in the corresponding institution. The **loose denotation** of a module $T$ is the class of models of the theory, i.e., $\mathrm{MOD}(T)$. The **tight denotation** of the module is the initial model of the theory, denoted $0_T$.

A module can have either loose of initial semantics, this is determined by the CafeOBJ conventions or else is directly specified by the user. CafeOBJ does not directly implement final semantics, however the loose semantics of behaviour modules uses final models in a crucial way (see [18, 8]).

Initial model semantics is available only for non-behaviour modules, and is supported by the following result:

**Theorem 1** Let $T$ be a theory in either MSA, OSA, RWL, or OSRWL. Then the initial model $0_T$ exists. $\Box$

This very important result appears in various variants and can be regarded as a classic of algebraic specification theory. The reader may wish to consult [23] for MSA, [22, 15] for OSA, [27] for RWL, and although, up to our knowledge, the result has not yet been published, we dont have any reasons to discard it for OSRWL.

Because of the importance of the construction of the initial model we briefly recall it here. Let $\Sigma$ be the signature of the theory consisting of a set $S$ of sorts (which is a partial order in the order-sorted case) and a ranked (by $S^*$) set of operation symbols (possibly overloaded). The $S$-sorted set $T_\Sigma$ of $\Sigma$-terms is the least $S$-sorted set closed under:

- each constant is a $\Sigma$-term ($\Sigma_{[],s} \subseteq T_{\Sigma,s}$), and

- $\sigma(t1 \dots tn) \in T_{\Sigma,s}$ whenever $\sigma \in \Sigma_{s1 \dots sn,s}$ and $ti \in T_{\Sigma,si}$ for $i \in \overline{1,n}$.

The operations in $\Sigma$ can be interpreted on $T_\Sigma$ in the obvious manner, thus making it into a $\Sigma$-algebra $0_\Sigma$. If $T$ is equational, then its ground part is a congruence $\equiv_T$ on $0_\Sigma$. Then $0_T$ is the quotient $0_\Sigma/_{\equiv_T}$, whose carriers are equivalence classes of $\Sigma$-terms under $\equiv_T$. If $T$ is a pure rewriting theory then $0_T$ is a rewriting logic model whose carriers $(0_T)_s$ are categories with $\Sigma$-terms as objects and *concurrent* rewrite sequences (using the rules of $T$) as arrows. Finally, rewrite theories including equations require the combination between the above two constructions.

The completeness of the operational semantics (which is mainly based on rewriting) is obtained via the completeness of the proof systems for equational logic [22], in one case, and for rewriting logic [27], in the second case.

# 5 Programming in the Large

In this section we are concerned with the semantics of the module interconnection system. CafeOBJ module interconnection system follows the principles of the OBJ module system which are inherited from earlier work on Clear [3]. Consequently our semantics is based on institutions employing the theory developed in [9]. In the actual case of CafeOBJ this institutional semantics is instantiated to the CafeOBJ cube, however its essential core can be presented at the level of institutions thus avoiding the particular details of the CafeOBJ cube logics.

## 5.1 Module Imports

Module imports constitute the primitive concept underlying the semantics of the CafeOBJ module interconnection system:

> **(Li)** A **module import** is a theory morphism between the imported and the importing module, and its **denotation** is the corresponding functor between the denotations of the modules.

In order to make this principle more precise let's consider a module import $T \trianglelefteq T'$ where $T$ is the imported module and $T'$ is the importing module. Let $\Im_T$ and $\Im_{T'}$ be the institutions of $T$, and $T'$ respectively. Then since $T'$ must inherit the logic underlying $T$, we have

$$\Im_T \sqsubseteq \Im_{T'}$$

**Definition 2** A **module import** $T \trianglelefteq T'$ is a global theory morphism which is also an inclusion.[4]
□

**Corollary 3** Module imports form a partial order. □

The following result defines the denotations of module imports, showing that for each model of the importing module we can "extract" the part corresponding to the imported module. So, given a module import $T \trianglelefteq T'$, this defines a functor $\_\restriction_T : \mathrm{MOD}(T') \to \mathrm{MOD}(T)$ in the following way:

**Proposition 4** Let $T \trianglelefteq T'$ be a module import and $(\Phi, \alpha, \beta)$ be the institution embedding $\Im_T \sqsubseteq \Im_{T'}$. The any $T'$-model $M'$ has a reduct to a $T$-model given by

[4] "Inclusion" here should be understood in the precise sense given by the *inclusion systems* of [9] which develops a categorical theory (further refined by [29]) generalising the set-theoretic concept of inclusion. In this case we deal with inclusions in the category of theory morphisms.

$$M'\restriction_T = \beta_{\Sigma'}(M')\restriction_\Sigma = \beta_{\overline{\Phi}(\Sigma)}(M'\restriction_{\overline{\Phi}(\Sigma)})$$

where $\Sigma$ is the signature of $T$ and $\Sigma'$ is the signature of $T'$. □

The reduct of $T'$-models to $T$-models is a two step process which can be done in two ways. One way is to first reduce the $T'$-model to the "less complex" institution $\Im_T$, then reduce the resulting model to the signature of $T$. Alternatively, we may first reduce the $T'$-model to the signature of $T$ (but mapped to $\Im_{T'}$) and then reduce the resulting model to the $\Im_T$. These show that when reducing models along module imports the two basic steps of reducing to a "simpler" paradigm and to a "smaller" signatures can be interchanged.

As with OBJ, CafeOBJ distinguishes between 3 basic kinds of imports. The CafeOBJ system supports only syntactic declarations for these different kinds of imports (see [26]), no other support is provided (in fact a full checking is undecidable). So, in order to avoid semantic inconsistencies (which at the end boil down to faulty specifications/programs) it is important to understand precisely at the level of the language semantics what kind of import one uses.

**Definition 5** An import $T \trianglelefteq T'$ is

- **protecting** iff $\mathrm{MOD}(T')\restriction_T = \mathrm{MOD}(T)$, i.e., for each $T$-model $M$ there exists a $T'$-model $M'$such that $M'\restriction_T = M$,
- **extending** iff for each $T$-model $M$ there exists a $T'$-model $M'$ and an inclusive model morphism $M \hookrightarrow M'\restriction_T$, and
- **using** otherwise.

□

This definition applies to both loose and initial semantics. In the case of initial semantics one has to consider the class of models of the theory consisting only of one model, i.e., the initial model. This can be achieved rigorously by using the *initial data constraints* of [14], which give an elegant way to restrict the class of models of a theory to only the initial model.

## 5.2 Parameterised Modules

Parameterised specification/programming is a very important feature of all languages in the OBJ family. The semantics of parameterised modules is based on the semantics of module imports since at the semantics level a parameterised module can be regarded as a special kind of module import (in which the parameter is imported).

> **(Lp)** A **parameterised module** $T[P]$, where $P$ is the parameter, is an import $P \trianglelefteq T$.
> A **view** (instantiating the parameter) is a global theory morphism $P \to P'$.

**Definition 6** Let $T[P]$ be a parameterised module and $v\colon\ P \to P'$ be a view. Let $\Im'$ be the least upper bound of $\Im_T$ and $\Im'_P$ in the CafeOBJ cube. Then the instantiation $T[v]$ is given by the following pushout in the category of $\Im'$-theories $\mathbb{T}h(\Im')$ (or, equivalently, in the category of global theory morphisms):

$$\begin{tikzcd} P \arrow[r, "\trianglelefteq"] \arrow[d, "v"'] & T \arrow[d] \\ P' \arrow[r] & T[v] \end{tikzcd}$$

□

This construction is supported by fundamental results showing that pushouts in the category of theories of an institution always exist provided pushouts for signatures exist [14, 9]. All CafeOBJ cube institutions have pushouts for signatures, however for the order-sorted case this can be non-trivial (see [25]).

The following result (see [9]) constitute the foundation for the semantics of parameter instantiation:

**Theorem 7** Let $T[P]$ be a parameterised module and $v\colon P \to P'$ be a view. Then $P' \to T[v]$ is a module import and $P' \trianglelefteq T[v]$ is protecting if $P \trianglelefteq T$ is protecting. □

### 5.3 Module Sum

Shared sum of modules (denoted as $+$) is one of the basic operations on modules.

**Definition 8** Given two modules $T$ and $T'$, $T + T'$ is the smallest theory such that $T \trianglelefteq T + T'$ and $T' \trianglelefteq T + T'$. □

**Corollary 9** $\Im_{T+T'} = \Im_T \sqcup \Im_{T'}$. □

This says that the institution of the sum unifies the paradigms of the institution of the components.

We can extend the basic result from [9] on sums of modules to:

**Proposition 10** Let $T$ and $T'$ be two modules. Then we have the following pushout-pullback square (in $\Im_T \sqcup \Im_{T'}$)

$$\begin{tikzcd} T \arrow[r, "\trianglelefteq"] & T + T' \\ T \cap T' \arrow[u, "\trianglelefteq"] \arrow[r, "\trianglelefteq"'] & T' \arrow[u, "\trianglelefteq"'] \end{tikzcd}$$

where $T \cap T'$ is the shared part (i.e., the intersection) of $T$ and $T'$. □

In [9] this result is used for deriving various properties of $+$, such that associativity, commutativity, etc.

The following result describes the semantics of $+$ by showing that any two consistent implementations of the components of the sum can be put together as an implementation of the sum of modules.

**Corollary 11** For any model $M$ of $T$ and any model $M'$ of $T'$ such that $M\restriction_{T\cap T'} = M'\restriction_{T\cap T'}$ there exists a unique model $M \oplus M'$ of $T + T'$ such that $(M \oplus M')\restriction_T = M$ and $(M \oplus M')\restriction_{T'} = M'$. □

### 5.4 Module Expressions

Module expressions are formed as iterations of the following basic constructs:

- imports,
- renamings,
- sum,
- (instantiations of) parameterised modules.

The evaluation of the module expressions results into a module that can be calculated as a colimit of theories in the style of Clear [3] in the least upper bound of the institutions of the basic constructs.

**Proposition 12** The denotation of the evaluation of a module expression is a limit of the denotations of the basic constructs. □

This result relies on a basic property of the CafeOBJ institutions called **exactness** (see [9]) for more detalis.

# 6 Programming in the Huge

Programming in the huge in CafeOBJ is based on Goguen's "hyperprogramming" approach [12], which is a semnatic based technique for the integration of diverse features of programming environments. This involves clusters of related text centered around a specification, plus module expressions which tell how to combine and transform such module clusters.

Technically, hyperprogramming employs techniques from programming in the large, such as evaluation of module expressions as co-limits (in this case in a suitable category of *module clusters*).

# 7 The Environment

The principal aim of the Cafe environment (an environment for CafeOBJ language) is to support the specification documents with formal contents. The emphasis is on "with formal contents". Specifically, a specification document contains (1) codes in formal specification in CafeOBJ (2) instructions of formal verification (execution via TRSs, theorem proving, etc.), and (3) the results of such verification. These contents ensure the rigour of specification, and allow the reviewer, inspector, browser, etc., to be convinced of its reliability.

But we would also like not to be fanatics. We would like to allow the user to insert informal explanations, in charts, in tables, in diagrams, and in native languages, as he so wishes. These explanations enhance legibility and usefulness of the documents.

To make the system friendly and to make system architecture modular and flexible, we take note of the overwhelming tide of networking practices. In particular, we observe that WWW browsers as front-ends enable the user to manipulate informations smoothly, and take advantage of network infrastructures fully. Specification documents should be available on networks, and be amenable to such manipulation.

The environment consists of roughly four parts:

**CafeOBJ interpreter.** In isolation, this part acts very much like the OBJ interpreter. It checks syntax and evaluates (reduces) terms. In fact, we already have a good interpreter. In this project, we shall enhance its performance. In particular, we construct an abstract TRS machine and a compiler, and incorporate them into the interpreter.

**Proof assistence system.** An interpreter may be well used as a theorem prover, but more powerful, dedicated provers are desirable. As proof engines, at least two kinds of inductive provers are considered. One is based on completion procedures, and the other on explicit structural induction. On top of these engines, we shall construct a proof assistence system that takes into account the particulars of CafeOBJ.

**Document manipulator.** This part takes care of every kinds of processing of specification documents over network. For one thing, it analyses specification documents to show the contents to the user (via WWW browsers, editors etc.), to extract instructions of evaluations and proofs, and to search for suitable documents in the libraries. For another, it manages documents on networks, retrieving, storing, caching them as requested.

**Specification libraries.** This part does not constitute the system per se, but is enhancing its usability. We do not intend to provide a comprehensive set of libraries, which is unrealistic. Rather, we are focusing on a couple of specific problem domains. We are now planning to establish libraries for object-oriented programming, database management systems, and interactive system.

# 8 Conclusions

We provided CafeOBJ with logical semantics based on institutions. Some of its main features are:

- simplicity and effectiveness via appropriate abstractness,
- cohesiveness,
- flexibility,
- provides support for multi-paradigm integration,
- provides support for the development of specification methodologies, and
- uses state-of-art methods in algebraic specification research.

The logical semantics constitute the mathematical basis of the CafeOBJ project, and it will play a guiding rôle in the future design decisions for CafeOBJ and in developing specification methodologies in Cafe. The planned "CafeOBJ Report" will synthesize the rôle played by this logical semantics for the CafeOBJ language.

## References


[1] Rod Burstall and Răzvan Diaconescu. Hiding and behaviour: an institutional approach. In A. William Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 75–92. Prentice-Hall, 1994. Also in Technical Report ECS-LFCS-8892-253, Laboratory for Foundations of Computer Science, University of Edinburgh, 1992.

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

[3] Rod Burstall and Joseph Goguen. The semantics of Clear, a specification language. In Dines Bjorner, editor, *Proceedings, 1979 Copenhagen Winter School on Abstract Software Specification*, pages 292–332. Springer, 1980. Lecture Notes in Computer Science, Volume 86; based on unpublished notes handed out at the Symposium on Algebra and Applications, Stefan Banach Center, Warsaw, Poland, 1978.

[4] Răzvan Diaconescu. *Category-based Semantics for Equational and Constraint Logic Programming*. PhD thesis, University of Oxford, 1994.

[5] Răzvan Diaconescu. Completeness of category-based equational deduction. *Mathematical Structures in Computer Science*, 5(1):9–41, 1995.

[6] Răzvan Diaconescu. A category-based equational logic semantics to constraint programming. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification*, volume 1130 of *Lecture Notes in Computer Science*, pages 200–221. Springer, 1996. Proceedings of 11th Workshop on Specification of Abstract Data Types. Oslo, Norway, September 1995.

[7] Răzvan Diaconescu. Category-based modularisation for equational logic programming. *Acta Informatica*, 33(5):477–510, 1996. To appear.

[8] Răzvan Diaconescu. Foundations of behavioural specification in rewriting logic. In *Proceedings, First International Workshop on Rewriting Logic and its Applications. Asilomar, California, September 1996.*, volume 5 of *ENTCS*. Elsevier Science, 1996. to appear.

[9] Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gerard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge, 1993. Proceedings of a Workshop held in Edinburgh, Scotland, May 1991.

[10] Kokichi Futatsugi, Joseph Goguen, Jean-Pierre Jouannaud, and Jose Meseguer. Principles of OBJ2. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, pages 52–66. ACM, 1985.

[11] Kokichi Futatsugi and Toshimi Sawada. Design considerations for cafe specification environment. In *Proc. OBJ2 10th Anniversary Workshop*, October 1995.

[12] Joseph Goguen. Hyperprogramming: A formal approach to software environments. In *Proceedings, Symposium on Formal Approaches to Software Environment Technology*. Joint System Development Corporation, Tokyo, Japan, January 1990.

[13] Joseph Goguen. *Theorem Proving and Algebra*. MIT, to appear 1996.

[14] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, January 1992.

[15] Joseph Goguen and Răzvan Diaconescu. An Oxford survey of order sorted algebra. *Mathematical Structures in Computer Science*, 4(4):363–392, 1994.

[16] Joseph Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Harmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification*, volume 785 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 1994.

[17] Joseph Goguen and Răzvan Diaconescu. An introduction to category-based equational logic. In V.S. Alagar and Maurice Nivat, editors, *Algebraic Methodology and Software Technology*, volume 936 of *Lecture Notes in Computer Science*, pages 91–126. Springer, 1995.

[18] Joseph Goguen and Grant Malcolm. A hidden agenda, 1996. draft, http://www-cse.ucsd.edu/recpubs.html.

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

[20] Joseph Goguen and José Meseguer. Models and equality for logical programming. In Hartmut Ehrig, Giorgio Levi, Robert Kowalski, and Ugo Montanari, editors, *Proceedings, 1987 TAPSOFT*, pages 1–22. Springer, 1987. Lecture Notes in Computer Science, Volume 250.

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

[22] Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992. Also, Programming Research Group Technical Monograph PRG–80, Oxford University, December 1989.

[23] Joseph Goguen, James Thatcher, and Eric Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. Technical Report RC 6487, IBM T.J. Watson Research Center, October 1976. In *Current Trends in Programming Methodology, IV*, Raymond Yeh, editor, Prentice-Hall, 1978, pages 80–149.

[24] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph Goguen, editor, *Algebraic Specification with OBJ: An Introduction with Case Studies*. Cambridge, to appear 1995. Also to appear as Technical Report from SRI International.

[25] Anne Elisabeth Haxthausen and Friederike Nickl. Pushouts of order-sorted algebraic specifications. In *Proceedings of AMAST'96*, 1996.

[26] http://www.ipa.go.jp/STC/CafeP/cafe.html. CafeOBJ user's manual.

[27] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, (93):73–155, 1992.

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

[29] Grigore Roşu and Virgil Căzănescu. Weak inclusion systems. *Mathematical Structures in Computer Science*, 1996. to appear.

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

[31] Andrzej Tarlecki, Rod Burstall, and Joseph Goguen. Some fundamental algebraic tools for the semantics of computation, part 3: Indexed categories. *Theoretical Computer Science*, 91:239–264, 1991. Also, Monograph PRG–77, August 1989, Programming Research Group, Oxford University.

# A Institutions

In this appendix we review some of the basic concepts and results on institutions, but also introduce some novel concepts dealing with the semantics of the multi-paradigm systems. A good introduction to institutions is [14], and [9] contains many results about institutions with direct application to modularisation.

From a logic perspective, institutions are much more abstract than Tarski's model theory, and also have another basic ingredient, namely signatures and the possibility of translating sentences and models across signature morphisms. A special case of this translation is familiar in first order model theory: if $\Sigma \to \Sigma'$ is an inclusion of first order signatures and $M$ is a $\Sigma'$-model, then we can form the *reduct* of $M$ to $\Sigma$, denoted $M\restriction_\Sigma$. Similarly, if $e$ is a $\Sigma$-sentence, we can always view it as a $\Sigma'$-sentence (but there is no standard notation for this). The key axiom, called the **satisfaction condition,** says that *truth is invariant under change of notation,* which is surely a very basic intuition for traditional logic.

**Definition 13** An **institution** $\Im = (\mathbb{S}ign, Sen, \mathrm{MOD}, \models)$ consists of

1. a category $\mathbb{S}ign$, whose objects are called **signatures,**
2. a functor $Sen\colon \mathbb{S}ign \to \mathbb{S}et$, giving for each signature a set whose elements are called **sentences** over that signature,
3. a functor $\mathrm{MOD}\colon \mathbb{S}ign \to \mathbb{C}at^{op}$ giving for each signature $\Sigma$ a category whose objects are called $\Sigma$-**models,** and whose arrows are called $\Sigma$-**(model) morphisms,** and
4. a relation $\models_\Sigma \subseteq |\mathrm{MOD}(\Sigma)| \times Sen(\Sigma)$ for each $\Sigma \in |\mathbb{S}ign|$, called $\Sigma$-**satisfaction,**

such that for each morphism $\varphi\colon \Sigma \to \Sigma'$ in $\mathbb{S}ign$, the **satisfaction condition**

$$M' \models_{\Sigma'} Sen(\varphi)(e) \text{ iff } \mathrm{MOD}(\varphi)(M') \models_\Sigma e$$

holds for each $M' \in |\mathrm{MOD}(\Sigma')|$ and $e \in Sen(\Sigma)$. We may denote the reduct functor $\mathrm{MOD}(\varphi)$ by $\_\restriction_\varphi$ and the sentence translation $Sen(\varphi)$ by $\varphi(\_)$. □

The following table shows the software engineering meaning of institution concepts for the case of specification languages.

| INSTITUTIONS | SPECIFICATION LANGUAGES |
|---:|---|
| signatures | syntactic declarations in modules |
| sentences | axioms in modules |
| models | (possible) implementations of modules |
| model morphisms | refinement between implementations |
| satisfaction relation | the implementation satisfies the axioms of the module |
| signature morphism | module import |
| sentence translation | importing the module axioms |
| model reduct | restricting the implementation of the importing module to an implemantation of the imported module |

**Definition 14** A **theory** $(\Sigma, E)$ in an institution $\Im = (\mathbb{S}ign, Sen, \mathrm{MOD}, \models)$ consists of a signature $\Sigma$ and a set $E$ of $\Sigma$-sentences closed under semantic entailment, i.e., $e \in E$ if $E \models_\Sigma e$.[5]

A **theory morphism** $\varphi\colon (\Sigma, E) \to (\Sigma', E')$ is a signature morphism $\varphi\colon \Sigma \to \Sigma'$ such that $\varphi(E) \subseteq E'$. Let $\mathbb{T}h(\Im)$ denote the category of all theories in $\Im$. □

[5]Meaning that $M \models_\Sigma e$ for any $\Sigma$-model $M$ that satisfies all sentences in $E$.

For any institution $\Im$, the model functor MOD extends to $\mathbb{T}h(\Im)$, by mapping a theory $(\Sigma, E)$ to the full subcategory $\mathrm{MOD}(\Sigma, E)$ of $\mathrm{MOD}(\Sigma)$ formed by the $\Sigma$-models that satisfy $E$.

Theories and theory morphisms have the following meaning in specification languages:

| INSTITUTIONS | SPECIFICATION LANGUAGES |
|---:|---|
| theory | module |
| theory morphism | module import |

**Definition 15** A theory morphism $\varphi\colon (\Sigma, E) \to (\Sigma', E')$ is **liberal** iff the reduct functor $\_\!\restriction_\varphi\colon \mathrm{MOD}(\Sigma', E') \to \mathrm{MOD}(\Sigma, E)$ has a left-adjoint $(\_)^\varphi$.

$$\begin{tikzcd}
M \models_\Sigma E & M \arrow[r] \arrow[d, "h"'] & (M^\varphi)\restriction_\varphi \arrow[dl, "h'"] & M^\varphi \arrow[dl, "\text{there exists a unique } h'"] \\
M' \models_{\Sigma'} E' & M'\restriction_\varphi & M' &
\end{tikzcd}$$

The institution $\Im$ is **liberal** iff each theory morphism is liberal. □

Liberality is a desirable property expressing the possibility of free constructions generalising the principle of "initial algebra semantics". General results [30] show that liberality is equivalent to the power of Horn axiomatisability.

Another very important property is related to the possibility of amalgamation of consistent implementations for different modules (for more details see [9]):

**Definition 16** An institution $\Im$ is **exact** iff the model functor $\mathrm{MOD}\colon \mathbb{S}ign \to \mathbb{C}at^{op}$ preserves co-limits. $\Im$ is **semi-exact** iff MOD preserves only pushouts. □

Semantics of some multi-paradigm systems might involve several different institutions which may be linked together by using the following concept:

**Definition 17** Let $\Im$ and $\Im'$ be institutions. Then an **institution morphism** $\Im \to \Im'$ consists of

1. a functor $\Phi\colon \mathbb{S}ign' \to \mathbb{S}ign$,
2. a natural transformation $\alpha\colon \Phi; Sen \Rightarrow Sen'$, and
3. a natural transformation $\beta\colon \mathrm{MOD}' \Rightarrow \Phi; \mathrm{MOD}$

such that the following **satisfaction condition** holds

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

for any $\Sigma'$-model $M'$ from $\Im'$ and any $\Phi(\Sigma')$-sentence $e$ from $\Im$.

If $\Phi$ admits a left-inverse left-adjoint (denoted as $\overline{\Phi}$) then we say that the institution morphism $\Im \to \Im'$ is an **embedding.** □

In the case of specification languages the components of instituion embeddings have the following meaning:

| INST. | SPECIFICATION LANGUAGES |
|---:|---|
| $\Phi$ | reduces the syntax of modules to syntax in a simpler paradigm |
| $\overline{\Phi}$ | regards the syntax of modules as (degenerated) syntax in a more complex paradigm |
| $\alpha$ | translates module axioms to axioms in a more complex paradigm |
| $\beta$ | extracts a simpler paradigm implementation from a module implementation |

The semantics of module imports for multi-paradigm systems requires a notion of morphism between theories belonging to different institutions. Such theory morphisms generalise the ordinary concept of theory morphism (Definition 14) in that it is "global" as opposed to "local".

**Definition 18** Let $(\Phi, \alpha, \beta): \Im \to \Im'$ be an institution morphism, and $T = (\Sigma, E)$ and $T' = (\Sigma', E')$ be theories in $\Im$ and $\Im'$ respectively. A **global theory morphism** $T \to T'$ is an $\Im$-signature morphism $\varphi: \Sigma \to \Phi(\Sigma')$ such that $\alpha_{\Sigma'}(\varphi(E)) \subseteq E'$. □

However in the case of institution embeddings we have an equivalent simpler formulation for global theory morphims. Recall from [14] that any institution embedding $(\Phi, \alpha, \beta): \Im \to \Im'$ gives rise to a functor $\mathbf{\Phi}: \mathbb{T}h(\Im) \to \mathbb{T}h(\Im')$ defined by

$$\mathbf{\Phi}(\Sigma, E) = (\overline{\Phi}, \alpha_{\overline{\Phi}(\Sigma)}(E)^\bullet)$$

**Proposition 19** Let $(\Phi, \alpha, \beta): \Im \to \Im'$ be an institution embedding and let $T \in \mathbb{T}h(\Im)$ and $T' \in \mathbb{T}h(\Im')$. Then a global theory morphism $T \to T'$ is the same with a $\Im'$-theory morphism $\mathbf{\Phi}(T) \to T'$. □

For readers familiar with indexed categories [31], the previous results just says that global theory morphisms are the arrows in the flattening (i.e., the Grothendick construction) of the indexed (by the category of institutions) category $\mathbb{T}h$.

# B Hidden Order Sorted Rewriting Logic

We devote this appendix to the (rather informal) presentation in some detail of HOSRWL (first introduced in [8] in the many sorted version) which embedds all CafeOBJ cube institutions. However, the deep understanding of HOSRWL requires further reading on its main components ([27] for RWL and [16, 18] for HSA) as well as their integration [8]. We assume familiarity with basic many sorted algebra which constitute the underlying level of all algebraic specification developments (relevant background can be found in [13, 23, 28]), but also with order sorted algebra [22, 15].

### Signatures

Let $D$ be a rewrite model for an order sorted signature (o.s. signature for short) $(V, \leq, \Psi)$.[6] A **hidden signature (over** $(V, \leq, \Psi)$**)** is a pair $(H, \leq, \Sigma)$, where $(H, \leq)$ is a partially ordered set of **hidden sorts,** disjoint from $V$, and $\Sigma$ is a $(H \cup V, \leq)$-o.s. signature, such that

(S1) each $\sigma \in \Sigma_{w,s}$ with $w \in V^*$ and $s \in V$ lies in $\Psi_{w,s}$, and

(S2) each $\sigma \in \Sigma_{w,s}$ has at most one element of $H$ in $w$.

If $w$ contains a hidden sort, the $\sigma \in \Sigma_{w,s}$ is called a **method** if $s \in H$ and an **attribute** if $s \in V$. Condition (S1) is a data encapsulation condition, and (S2) says that methods and attributes act on (states of) single objects.

A **hidden rewrite signature** is given by $(H, \leq, \Sigma, E)$ where $(H, \leq, \Sigma)$ is a hidden o.s. signature over $(V, \leq, \Psi)$, and $E$ is a collection of $\Sigma$-equations.

A **hidden sorted rewrite signature morphism** $\phi: (H, \leq, \Sigma, E) \to (H', \leq, \Sigma', E')$ is an o.s. signature morphism $(H \cup V, \leq, \Sigma) \to (H' \cup V, \leq, \Sigma')$ such that

(M1) $\phi(v) = v$ for all $v \in V$ and $\phi(\sigma) = \sigma$ for all $\sigma \in \Psi$,

(M2) $\phi(H) \subseteq H'$ (i.e., hidden sorts are mapped to hidden sorts),

(M3) if $\sigma' \in \Sigma'_{w',s'}$ and some sort in $w'$ lies in $H'$, then $\sigma' = \phi(\sigma)$ for some $\sigma \in \Sigma$,

(M4) if $\phi(h) < \phi(h')$ for any hidden sorts $h, h' \in H$, then $h < h'$, and

[6] This is refered as the **signature of data,** while $D$ is called the **model of data.**

(M5) $\phi(E) \models_{\Sigma'} E'$.

The first two conditions say that hidden sorted signature morphisms preserve visibility and invisibility for both sorts and operations, the third and fourth conditions express the encapsulation of classes and subclassesm (in the sense that no new methods or attributes can be defined on an imported class), while the fifth expresses the encapulation of structural axioms.

### Sentences

Given a signature $(H, \leq, \Sigma, E)$, a sentence is either a (possibly conditional) **equation** (modulo $E$) or else a (possibly conditional) **rule** (modulo $E$). Since equations are very traditional to algebraic specification, we concentrate here on rules. A conditional rule is written as

$$(\forall X)\ [t] \rightarrow [t']\ \textbf{if}\ [u_1] \rightarrow [v_1] \ldots [u_k] \rightarrow [v_k]$$

where $t, t', u_i, v_i$ are $\Sigma$-terms with variables $X$ and modulo the equations in $E$ (i.e., equivalence classes of $\Sigma$-terms modulo the congruence determined by $E$). The left-hand side of **if** is the head of the rule and the right-hand side is the condition of the rule.

Given a signature morphism $\phi\colon (H, \leq, \Sigma, E) \rightarrow (H', \leq, \Sigma', E')$ the translation of sentences is defined by the translation of $\Sigma$-terms (modulo $E$) to $\Sigma'$-terms modulo $E'$ along $\phi$ by replacing all symbols in $\Sigma$-terms with the corresponding symbols for $\Sigma'$. Condition (M5) enforces the correctness of this definition. For a full rigorous treatment of this issue the reader is advised to consult [4, 7].

### Models

Given an algebraic theory $(\Sigma, E)$, a **rewrite model** for $(\Sigma, E)$ is given by the interpretation of the algebraic theory into $\mathbb{C}at$. More concretely, a model $M$ interprets each sort $s$ as a category $M_s$, and each operation $\sigma \in \Sigma_{w,s}$ as a functor $\sigma_M\colon M_w \rightarrow M_s$, where $M_w$ stands for $M_{s_1} \times \ldots \times M_{s_n}$ for $w = s_1 \ldots s_n$. Each $\Sigma$-term $t\colon w \rightarrow s$ gets a functor $t_M\colon M_w \rightarrow M_s$ by evaluating it for each assignment of the variables occuring in $t$ with arrows from the corresponding carriers of $M$. The satisfaction of an equation $t = t'$ by $M$ is given by $t_M = t'_M$;[7] in particular all structural equations should be satisfied by $M$. A model morphism is a family of functors indexed by the sorts commuting the interpretations of the operations in $\Sigma$.

This algebra "enriched" over $\mathbb{C}at$ is a special case of *category-based equational logic* (see [4, 5, 17]) when letting the category $\mathbb{A}$of models to be the interpretations of $\Sigma$ into $\mathbb{C}at$ as abovely described, the category $\mathbb{X}$ of domains to be the category of many sorted sets, and the forgetful functor $\mathcal{U}\colon \mathbb{A} \rightarrow \mathbb{X}$ forgetting the interpretations of the operations and the composition between the arrows, i.e., mapping each category to its set of arrows. This enables the use of the machinery of category-based equational logic as a technical aide to the model theory of RWL.

A **hidden sorted rewrite model** $M$ for a hidden sorted rewrite signature $(H, \Sigma, E)$ over $(V, \Psi, D)$ is just a $(\Sigma, E)$-rewrite model for such that $M\restriction_{\Psi} = D$.

### Satisfaction

Let $(H, \leq, \Sigma, E)$ be a hidden sorted signature, $[\rho]$ be a sentence,[8] and $M$ be a model for this signature. Satisfaction in RWL of ordinary equations was explained in the paragraph on sentences, so we concentrate on the satisfaction of rules.

The satisfaction of a rewrite rule $(\forall X)\ [t] \rightarrow [t']\ \textbf{if}\ [u_1] \rightarrow [v_1] \ldots [u_k] \rightarrow [v_k]$ by $M$ has a rather sophisticated definition using the concept of *subequaliser*. Let $w$ be the string of sorts associated to the collection of variables $X$. Then

[7] This definition extends without difficulty to conditional equations.

[8] We extend the equivalence class notation from terms to sentences in the obvious way.

$$M \models (\forall X)\ [t] \to [t']\ \textbf{if}\ [u_1] \to [v_1] \ldots [u_k] \to [v_k]$$

iff there exists a natural transformation $J_M; t_M \Rightarrow J_M; t'_M$ where $J_M\colon\ Subeq((u_{iM}, v_{iM})_{i \in \overline{1,k}}) \to M_w$ is the subequaliser functor, i.e., the functor component of the final object in the category having pairs $(Dom(S) \xrightarrow{S} M_w, (S; u_{iM} \overset{\alpha_i}{\Rrightarrow}, S; v_{iM})_{i \in \overline{1,k}})$ as objects and functors $H$ such that $H; S' = S$ and $H\alpha' = \alpha$ as arrows.

The satisfaction in HOSRWL is **behavioural** (denoted by $\models\!\equiv$) and is defined as

$$M \models\!\equiv [\rho]\ \text{iff}\ \overline{M} \models [\rho]$$

where $\overline{M}$ is the **behaviour image**[9] of $M$ obtained by factoring the unique homomorphism from $M$ to the final model.[10] Informally, $\overline{M}$ identifies all elements and transitions that are "observationally the same", but also adds new "observational transitions".

A proof of the following result (but for the many sorted case) can be found in [8]:

**Theorem 20 [SATISFACTION CONDITION FOR HOSRWL]** Let $\phi\colon\ (H, \leq, \Sigma, E) \to (H', \leq, \Sigma', E')$ be a morphism of hidden sorted rewrite signatures, $M'$ be a $(H', \leq, \Sigma', E')$-rewrite model, and $\rho$ be a $\Sigma$-rule or a $\Sigma$-equation. Then

$$M' \models\!\equiv_{(H', \leq, \Sigma', E')} \phi([\rho])\ \text{iff}\ M'\restriction_\phi \models\!\equiv_{(H, \leq, \Sigma, E)} [\rho]$$

□

[9] See [8] for the formal definition.

[10] However in case of operations with visible arguments and hidden sort, the final modle might not exist, but we can instead use the final model in the signature without these operations. For more details see [8].

# Contents