Logical Semantics for CafeOBJ

Răzvan Diaconescu* and Kokichi Futatsugi

Japan Advanced Institute for Science and Technology

diacon@stoilow.imar.ro, kokichi@jaist.ac.jp

Abstract

This paper presents the semantics of CafeOBJ system and language. CafeOBJ is a succesor of the famous algebraic specification and programming language OBJ but adding several new primitive paradigms to the traditional OBJ language, such as rewriting logic, and behavioural concurrent specification.

We do not address here the detalied mathematical aspects of CafeOBJ semantics (which sometimes could be rather sophisticated), but rather survey this semantics. However all concepts and facts are fully backed by solid mathematical concepts and results, and we give pointers to mathematical works backing our claims. We provide in the appendices very brief surveys of several key structures; we hope this will make this paper more self contained. Some familiarity with the OBJ tradition (including some key mathematical concepts) is of course necessary for understanding the semantics of CafeOBJ.

1 Introduction

1.1 Principles of CafeOBJ semantics

CafeOBJ is a declarative language in the same way other OBJ-family languages (OBJ, Eqlog [23, 5], FOOPS [25], Maude [33]) 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 [24].

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

- programming "in the small",

*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 [18]) rather than in the traditional sense.

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

- 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 precise mathematical semantics, programming in the huge can be approached semantically using some general techniques developed for programming in the large (see [16]), the environment cannot (and should not) be given formal precise 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:

> **(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.

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 [11].

Equational specification and programming

This is inherited from OBJ [28, 15] 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*. We do not insist here on this feature since this is extensively covered by the already vast OBJ related literature.

Rewriting logic specification

This is based on a simplified version of Meseguer's *rewriting logic* [33] (abbreviatted **RWL**) specification framework for concurrent system which gives a non-trivial extension of traditional algebraic specification towards concurrency. RWL incorporates many different models of concurrency in a natural, simple, and elegant way, thus giving CafeOBJ a wide range of applications. Unlike Maude [31], the current CafeOBJ design does not fully support *labelled* RWL which permits full reasoning about multiple transitions between states, but provides proof support for reasoning about the *existence* of transitions between states of concurrent systems via a built-in predicate with dynamic definition encoding both the proof theory of RWL and the user defined transitions (rules) into equational logic.

From a methodological perspective, CafeOBJ emphasises the use of RWL transitions for *declarative encoding of algorithms* [13].

Behavioural specification

Behavioural specification [20, 22] 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 new form of abstraction can be very powerful in the specification or verification of software systems since it naturally embedds other useful paradigms such as concurrency, object-orientation, constraints, non-determinism, etc. (see [22] for details). Behavioural abstraction 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, which also generalises process algebra and transition systems (see [22]).

CafeOBJ directly supports behavioural specification and its proof theory through special language constructs, such as hidden sorts, behavioural operations and behavioural axioms (stating behavioural satisfaction). The advanced coinduction proof method receives some support in CafeOBJ via a default coinduction relation. In CafeOBJ coinduction can be used either in the classsical HSA sense [22] for proving behavioural equivalence of states of objects, or for proving behavioural transitions (which appear when applying behavioural abstraction to RWL; see [11, 7] for more details).

Object orientation

In CafeOBJ there are (at least) two sources of object-orientation. The first is given by the rewriting logic á la Maude treatment of objects which is implementation oriented, the second one is given by the behavioural specification of objects which is more faithful to the principle of state encapsulation. CafeOBJ does not make any specific choice on the kind of object-orientation since it regards the object-orientation as a derived feature rather than a primary paradigm of the language. This gives the user the freedom to chose the most suitable form of object-orientation for their applications. However, the current CafeOBJ methodologies strongly emphasise the treatment of concurrent objects within behavioural specification [13, 29].

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. However, the theory supporting the CafeOBJ module system represents an updating of the original Clear/OBJ concepts to the more sophisticated situation of multi-paradigm systems involving theory morphisms accross institution embeddings [12], and the concrete design of the language revise the traditional OBJ view on importation modes and parameters.

Powerful type system

CafeOBJ type system allows subtypes and is based on *order sorted algebra* [26, 19]. This provides a mathematically rigorous form of runtime type checking and error handling, giving 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 [19].

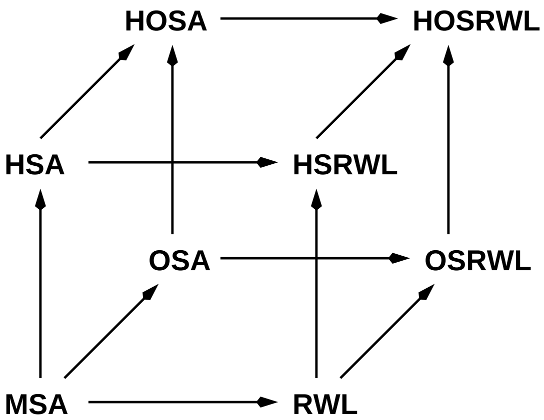
Since at the level of logical semantics the type system is orthogonal to the other features, the CafeOBJ designers decided to keep the concrete OSA formalism relatively open at the level of the language definition. A favoured solution appear to use Meseguer's *membership equational logic* [32] in the background with a revised version of Goguen's OSA [19] as user interface.

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	[17]
OSA	order sorted algebra	algebraic specification with subtypes	[17, 26, 19]
HSA	hidden sorted algebra	behavioural concurrent specification	[20]
HOSA	hidden order sorted algebra	behavioural specification with subtypes	[20, 1]
RWL	rewriting logic	rewriting logic specification	[33]
OSRWL	order sorted rewriting logic	rewriting logic specification with subtypes	
HSRWL	hidden sorted rewriting logic	behavioural rewriting logic specification	[11, 7]
HOSRWL	hidden order sorted rewriting logic	behavioural rewriting logic 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 [18, 14]; see Appendix A and [12] for more details), and which are shown by the following CafeOBJ **cube** (the orientation of arrows correspond to embedding “less complex” into “more complex” logics).



More rigorously, when dealing with pre-defined data types, the semantics must involve *constraint logics* [5, 8]. For example, the Boolean pre-defined type plays a special rôle in CafeOBJ since conditions of conditional equations or transitions (rules) are just Boolean terms. Also, semantics of libraries can be elegantly solved with constraint logic. 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 [8]), 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]

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. Notice that $\sqcup$ and $\sqcap$ form a lattice structure due to the symmetry of the cube.

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 to *basic specifications*, 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 specifications,
- rewriting specifications,
- behavioural specifications, and
- behavioural rewriting specifications.

The membership of a basic specification to a certain class is determined by the CafeOBJ convention that each basic specification should be regarded as implementing the simplest possible combination of paradigms resulting form its syntactic content (see [35]).

4.1 Loose vs. Tight Semantics

The key concept of programming in the small is the *satisfaction relation* between the models and the sentences of a given specification, which is also the key notion of institutions. Each kind of specification has its own concept of satisfaction, and Appendix B surveys them very briefly.

The table of Section 3 also shows the underlying logic corresponding to each class of basic specifications. Since specifications can be regarded as finite sets of sentences in the underlying logic, this enables us to formulate the principle of semantics of CafeOBJ programming in the small:

> **(S)** We identify between basic specifications and theories generated in the corresponding institution. Let $[\![T]\!]$ be the **denotation** of a basic specification T. Then $[\![T]\!]$ is the class of models of the theory, i.e., $\mathrm{MOD}(T)$, if **loose**, and it is the initial model 0_T of the theory, if **tight**.

A basic specification can have either loose of initial semantics, and this can be directly specified by the user. CafeOBJ does not directly implement final semantics, however the loose semantics of behavioural specifications uses final models in a crucial way (see [22, 11]).

[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.

Initial model semantics is available only for non-behaviour specification, 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 [27] for MSA, [26, 19] for OSA, [33] 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 Σ 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_Σ of Σ-terms is the least S-sorted set closed under:

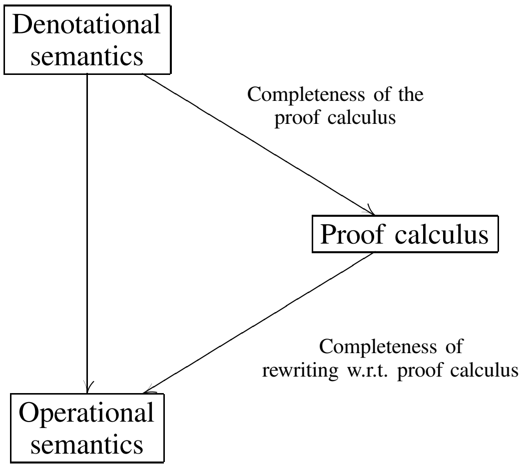
- each constant is a Σ-term ($\Sigma_{[],s} \subseteq T_{\Sigma,s}$), and
- $\sigma(t1 \ldots tn) \in T_{\Sigma,s}$ whenever $\sigma \in \Sigma_{s1\ldots sn,s}$ and $ti \in T_{\Sigma,si}$ for $i \in \overline{1,n}$.

The operations in Σ can be interpreted on T_Σ in the obvious manner, thus making it into a Σ-algebra 0_Σ. If T is equational, then its ground part is a congruence $\equiv_T$ on 0_Σ. Then 0_T is the quotient $0_\Sigma/_{\equiv_T}$, whose carriers are equivalence classes of Σ-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 Σ-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.

4.2 Operational vs. Logical Semantics

The operational semantics is responsible for the (correct) execution of specifications/ programs. As with OBJ, the CafeOBJ operational semantics is based on rewriting, which in the case of proofs is used without directly involving the user defined transitions (rules) but rather involving them via the built-in semantic transition predicate. For executions of concurrent systems specified in rewriting logic, CafeOBJ uses both the user-defined transitions and equations.

The completeness of the operational semantics with respect to the logical semantics is a two-layer completeness going via the important intermediate level of the proof calculi.



The completeness of the proof calculus is one of the most important class of results in algebraic specification, for equational logic we refer to [26], and for rewriting logic to [33]. In the case of rewriting logic the relationship between the proof calculus and rewriting is very intimate, but for equational logic the completeness of rewriting can be found, among other many places, in [17, 10].

5 Programming in the Large

In this section we survey 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 by extending the theory developed in [14] to multi-paradigm systems based on several institutions. 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. The following principle governs the semantics of programming in-the-large in CafeOBJ:

> **(L)** We identify between the structured specifications and the theories corresponding to flattening them to basic specifications. The structuring constructs are modelled by (extra) theory morphisms between the corresponding theories. The denotation $[\![T]\!]$ of a structured specification is determined from the denotations of the components recursively via the structuring constructs involved.

The general structuring mechanism is constituted by *module expressions*, which are iterations of several basic structuring operations, such as (multiple) imports, parameters, instantiation of parameters by views, translations, etc. In this section we discuss the most important ones: imports and parameterization.

5.1 Module Imports

Module imports constitute the primitive concept underlying any module interconnection systems, here is its mathematical definition:

Definition 2 A **module import** $T \trianglelefteq T'$ is an inclusion extra theory morphism $T \hookrightarrow T'$. $\Box$

Notice that in the previous definition, the institution $\Im_{T'}$ of T' embeds the institution $\Im_T$ of T (i.e., $\Im_T \sqsubseteq \Im_{T'}$ in the fixed lattice of the CafeOBJ cube institutions). The (weak) inclusion system underlying this definition is that introduced by Corollary 28.

By following the OBJ tradition, we can distinguish between 3 basic kinds of imports, **protecting**, **extending**, and **using**. At the level of the language, these should be treated just as semantic declarations which determine the denotation of the importing module from the denotation of the imported module.

Definition 3 Fix an import $T \trianglelefteq T'$. Then $[\![T']\!] =$

- $\{M' \mid M' \models T',\ M'$ protecting (and free if T' is initial) expansion of a model $M \in [\![T]\!]\}$, when the importation mode is *protecting*,

- $\{M' \mid M' \models T',\ M'$ extending (and free if T' is initial) expansion of a model $M \in [\![T]\!]\}$, when the importation mode is *extending*, and
- $\{M' \mid M' \models SP'$ and M' is *using* (and free if SP' has initial denotation) expansion of some model $M \in [\![SP]\!]\}$, when the importation mode is *using*.

$\Box$

Multiple imports are handled by a lattice structure on inclusions (see [14, 4].

Definition 4 Given two modules T and T', their **shared part** $T \wedge T'$ is the *greatest lower bound* in the lattice of imports $\trianglelefteq$ and their **sum** $T + T'$ is the *lowest upper bound.* $\Box$

We can easily notice the that the institution of the sum unifies the paradigms of the institution of the components, i.e., $\Im_{T+T'} = \Im_T \sqcup \Im_{T'}$.

The following extends a basic result on multiple imports from [14] to the multi-paradigm case:

Corollary 5 Let T and T' be two modules. Then we have the following pushout-pullback square (in $\Im_T \sqcup \Im_{T'}$)

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

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

5.2 Parameterization

Parameterized specification/programming is a very important feature of all modern declarative languages. The mathematical definition of parameterized modules is based on the concept of *injection* as composite between an inclusion and an isomorphism.

Definition 6 A **parameterized specification (module)** $T(X :: P)$ is an *injection* (extra theory morphim) $P \xrightarrow{X} T$. A **view** is just an extra theory morphism. $\Box$

Notice that the institution of the parameter P is embedded in the institution of the body T (i.e., $\Im_P \sqsubseteq \Im_T$). The denotation $[\![T]\!]$ of the body is determined from the denotation of the parameter accordingly to the parameterization mode as in the case of importing modules.

We distinguish two basic approaches on parameters: a "shared" and a "non-shared' one. In the "non-shared" approach, the multiple parameters are mutually disjoint (i.e., $\mathrm{Im}(X) \wedge \mathrm{Im}(X') = \emptyset$ for X and X' two different parameters) and they are also disjoint from any module imports $T_0 \trianglelefteq T$ (i.e., $\mathrm{Im}(X) \wedge T_0 = \emptyset$). In the "shared" approach this principle is relaxed to being disjoint *outside common imports*, i.e., $\mathrm{Im}(X) \wedge \mathrm{Im}(X') = \sum_{T_1 \trianglelefteq X} T_1 \wedge \sum_{T_1 \trianglelefteq X'} T_1$ for X and X' two different parameters and $\mathrm{Im}(X) \wedge T_0 = \sum_{T_1 \trianglelefteq X} \wedge T_0$ for all $T_0 \trianglelefteq T$. The "non-shared" approach has the potentiality of a much more powerful module system, while the "shared" approach seems to be more convenient to implement. The CafeOBJ definition contains both of them, for details on "non-shared" vs. "shared" parameterization and for a more detailed presentation of a module system based on this theory, see [13].

Definition 7 Let $T(X :: P)$ be a parameterized module and $v\colon P \to P'$ be a view. Then the instantiation $T(v)$ is given by the following pushout in the sense of the main co-limit theorem of [12]

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

in the "non-shared" approach and by the following co-limit in the sense of the main co-limit theorem of [12]

$$\begin{tikzcd} P' \wedge T \arrow[drr, "\unlhd"] \arrow[ddr, "\unlhd"'] & & \\ & P \arrow[r, "X"'] \arrow[d, "v"'] & T \arrow[d, "v'"] \\ & P' \arrow[r, "\unlhd"'] & T(v) \end{tikzcd}$$

in the "shared" approach. In both cases the embedding institution is $\Im_T \sqcup \Im_{P'}$. $\Box$

6 Programming in the Huge

Programming in the huge in CafeOBJ is based on Goguen's "hyperprogramming" approach [16], which is a semantic 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 assistance 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.

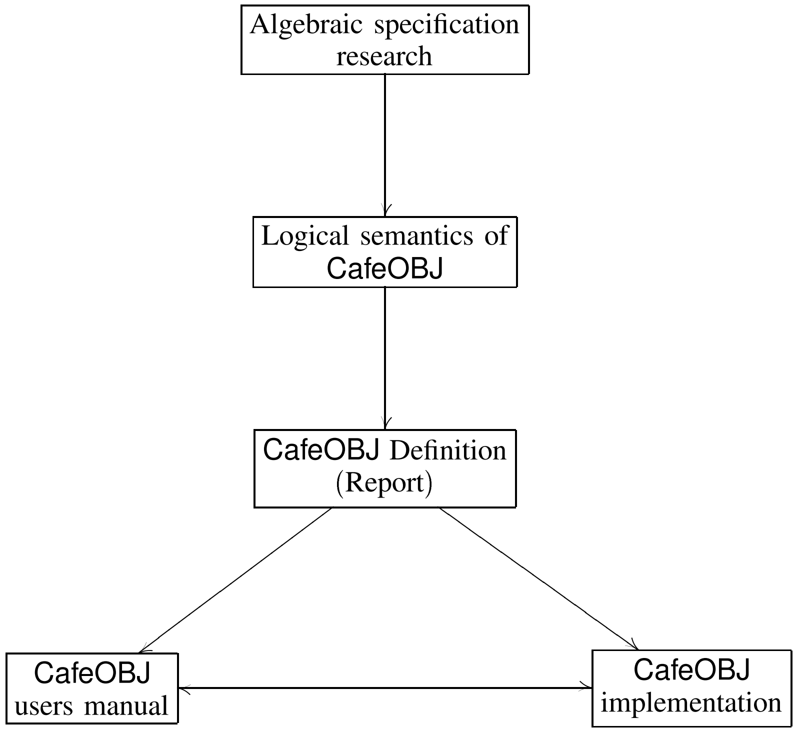
Specification libraries. This part does not constitute the system per se, but is enhancing its usability. We do not intend to provide a comprehensive set of libraries, which is unrealistic. Rather, we are focusing on a couple of specific problem domains. We are now planning to establish libraries for 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 constitutes the mathematical basis of the CafeOBJ project by putting together bits and pieces of modern algebraic specification research. More precisely, the Logical Semantics constitutes the origin of the concrete definition of CafeOBJ which is presented in the CafeOBJ Report [13]. The CafeOBJ Report plays the rôle of the language reference document and guiding the various CafeOBJ implementations and their manuals, as shown in the following figure.


Algebraic specification research
Logical semantics of CafeOBJ
CafeOBJ Definition (Report)
CafeOBJ users manual
CafeOBJ implementation

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] Virgil Emil Căzănescu and Grigore Roşu. Weak inclusion systems. *Mathematical Structures in Computer Science*, 7(2), 1997.

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

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

[7] Răzvan Diaconescu. Behavioural rewriting logic: semantic foundations and proof theory, October 1996. Submitted to publication.

[8] 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.

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

[10] Răzvan Diaconescu. Completeness of semantic paramodulation: a category-based approach. Technical Report IS-RR-96-0006S, Japan Advanced Institute for Science and Technology, 1996.

[11] 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 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1996.

[12] Răzvan Diaconescu. Extra theory morphisms for institutions: logical semantics for multi-paradigm languages. *Jour. of Applied Categorical Structures*, 1998. To appear; a preliminary version appeared as JAIST Technical Report IS-RR-97-0032F in 1997.

[13] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998. To appear.

[14] 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.

[15] 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.

[16] 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.

[17] Joseph Goguen. *Theorem Proving and Algebra*. MIT, To appear.

[18] 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.

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

[20] 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.

[21] 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.

[22] Joseph Goguen and Grant Malcolm. A hidden agenda. Technical Report CS97-538, University of California at San Diego, 1997.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Shusaku Iida, Michihiro Matsumoto, Răzvan Diaconescu, Kokichi Futatsugi, and Dorel Lucanu. Concurrent object composition in CafeOBJ. Technical Report IS-RR-98-0009S, Japan Advanced Institute for Science and Technology, 1998. Submitted to publication.

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

[31] P. Lincoln M. Clavel, S. Eker and J. Meseguer. Principles of Maude. In *Proceedings, First International Workshop on Rewriting Logic and its Applications. Asilomar, California, September 1996.*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1996.

[32] José Meseguer. Membership algebra as a logical framework for equational specification. Invited paper at the Workshop on Algebraic Development Techniques, Tarquinia 1997.

[33] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[34] 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.

[35] Ataru Nakagawa, Toshimi Sawada, and Kokichi Futatsugi. CafeOBJ user manual, 1997. Version 1.3, `ftp://sra.co.jp/pub/lang/CafeOBJ/Manual/`.

[36] 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.

[37] Andrzej Tarlecki. Moving between logical systems. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification*, Lecture Notes in Computer Science, pages 478–502. Springer, 1996. Proceedings of 11th Workshop on Specification of Abstract Data Types. Oslo, Norway, September 1995.

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 [18], and [14] contains many results about institutions with direct application to modularization. The reference for the new mathematics for multi-paradigm systems is [12].

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 Σ'-model, then we can form the *reduct* of M to Σ, denoted $M\restriction_\Sigma$. Similarly, if e is a Σ-sentence, we can always view it as a Σ'-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 8 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 Σ a category whose objects are called **Σ-models**, and whose arrows are called **Σ-(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 $\varphi(\_)$. $\Box$

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 implementation of the imported module

Definition 9 Let $\Im = (\mathbb{S}ign, Sen, \mathrm{MOD}, \models)$ be an institution. For any signature Σ the closure of a set E of Σ-sentences is $E^\bullet = \{e \mid E \models_\Sigma e\}$[4]. (Σ, E) is a **theory** iff $E = E^\bullet$.

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

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 $\Im$, and $sign^{\Im}$ the forgetful functor $\mathbb{T}h(\Im) \to \mathbb{S}ign$. $\Box$

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

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

INSTITUTIONS	SPECIFICATION LANGUAGES
theory	(flattened) module
theory morphism	module import, view, module parameter

Liberality is a desirable property expressing the possibility of free constructions generalizing the principle of "initial algebra semantics". General results [36] show that liberality is equivalent to the power of Horn axiomatizability.

Definition 10 A theory morphism $\varphi\colon (\Sigma,E) \to (\Sigma',E')$ is **liberal** iff the reduct functor $\_\restriction_\varphi\colon \textsc{Mod}(\Sigma',E') \to \textsc{Mod}(\Sigma,E)$ has a left-adjoint $(\_)^\varphi$.

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

The institution $\Im$ is **liberal** iff each theory morphism is liberal. When (Σ,E) is the "empty" theory for the signature Σ, we denote $(\_)^\varphi$ by $\_/E'$. $\Box$

Another very important property expresses the possibility of amalgamation of consistent implementations for different modules (for more details see [14]); the following is its formulation within the framework of institutions:

Definition 11 An institution $\Im$ is **exact** iff the model functor $\textsc{Mod}\colon \mathbb{S}ign \to \mathbb{C}at^{op}$ preserves co-limits. $\Im$ is **semi-exact** iff $\textsc{Mod}$ preserves only pushouts, and **weakly semi-exact** iff $\textsc{Mod}$ maps pushouts to weak[5] pushouts. $\Box$

Semantics of multi-paradigm systems involves several different institutions which have to be linked together by using the following concept:

Definition 12 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 $\beta\colon \textsc{Mod}' \Rightarrow \Phi; \textsc{Mod}$

[5] See [30] for the definition and discussion of the concept of *weak* universal properties.

such that the following **satisfaction condition** holds

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

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

In the literature there are several concepts of institution morphism, each of them being adequate to some specific class of problems. A good survey of various concepts of institution morphism discussing their usefulness can be found in [37]. The definition presented above and originally given by Goguen and Burstall [18] seems to be the most adequate for our approach. However, for obtaining some technical properties for extra theory morphisms, some technically stronger versions of this institution morphism are needed. These are very natural technical conditions which are easily satisfied in practice. The following definition is taken from [12]:

Definition 13 An institution morphism $(\Phi, \alpha, \beta) : \mathfrak{I} \to \mathfrak{I}'$ is

- **[strong] embedding** iff Φ admits a [left-inverse] left-adjoint $\overline{\Phi}$,
- **liberal** iff $\beta_{\Sigma'}$ has a left-adjoint $\overline{\beta}_{\Sigma'}$ for each $\Sigma' \in |\mathbb{S}ign'|$, and **persistent** iff in addition $\overline{\beta}_{\Sigma'}$ are left-inverses to $\beta_{\Sigma'}$ too, and
- **[weakly] additive** iff the squares defining the naturality of β are [weak] pullbacks.

$$\begin{tikzcd}
\mathrm{Mod}(\Phi(\Sigma')) & \mathrm{Mod}'(\Sigma') \arrow[l, "\beta_{\Sigma'}"'] \\
\mathrm{Mod}(\Phi(\Sigma'_1)) \arrow[u, "\mathrm{Mod}(\Phi(\varphi))"] & \mathrm{Mod}'(\Sigma'_1) \arrow[l, "\beta_{\Sigma'_1}"] \arrow[u, "\mathrm{Mod}'(\varphi)"']
\end{tikzcd}$$

□

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

INST.	SPECIFICATION LANGUAGES
Φ	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
α	translates module axioms to axioms in a more complex paradigm
β	extracts a simpler paradigm implementation from a module implementation

Extra Theory Morphisms

Extra theory morphisms generalize the ordinary concept of theory morphism (Definition 9) in that it maps theories across an institution morphism. Intra (i.e., ordinary) theory morphisms can be regarded as special cases when the institution morphism is an identity. In this section we give a summary of the main definitions and results about extra theory morphisms supporting the semantics of structuring CafeOBJ specifications; all definitions and results of this section are taken from [12].

Definition 14 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 **extra theory morphism** $T \to T'$ is an $\Im$-signature morphism $\varphi \colon \Sigma \to \Phi(\Sigma')$ such that $\alpha_{\Sigma'}(\varphi(E)) \subseteq E'$. $\Box$

Fact 15 Any institution embedding $(\Phi, \alpha, \beta) : \Im \to \Im'$ gives rise to a functor $\Phi^* : \mathbb{T}h(\Im) \to \mathbb{T}h(\Im')$ defined by

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

where ζ is the unit of the adjoint pair of functors $\Phi, \overline{\Phi}$. $\Box$

Proposition 16 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 there is a natural bijection between extra theory morphisms $T \to T'$ and $\Im'$-theory morphisms $\Phi^*(T) \to T'$. $\Box$

Model reducts are the semantic aspect of theory morphisms, therefore they play a central rôle in any semantics based on institutions. Model reducts for extra theory morphisms generalize ordinary model reducts for intra theory morphisms; they are introduced by the following result which can also be regarded as a Satisfaction Condition for extra theory morphisms:

Proposition 17 Let $(\Phi, \alpha, \beta) : \Im \to \Im'$ be an institution morphism. For any extra theory morphism $\varphi \colon (\Sigma, E) \to (\Sigma', E')$ there is a **reduct functor** $\_\!\upharpoonright_\varphi \colon \textsc{Mod}(T') \to \textsc{Mod}(T)$ defined by

$$M' \upharpoonright_\varphi = \beta_{\Sigma'}(M') \upharpoonright_\varphi$$

for M' any (Σ', E')-model. If (Φ, α, β) is an embedding, then

$$M' \upharpoonright_\varphi = \beta_{\Sigma\overline{\Phi}}(M' \upharpoonright_{\varphi'}) \upharpoonright_{\zeta_\Sigma}$$

where $\varphi' : \overline{\Phi}(\Sigma) \to \Sigma'$ is the free extension of $\varphi \colon \Sigma \to \Phi(\Sigma')$. $\Box$

The concept of *model expansion* is dual to model reducts, and plays a crucial rôle for defining the denotations of structured specifications:

Definition 18 Given an extra theory morphism $\varphi \colon T \to T'$, and a model M of T, an **expansion of** M **along** φ is a model M' of T' satisfying the following properties:

- $M' \upharpoonright_\varphi = M$ iff the expansion is **protecting**,
- there is an *injective*[6] model homomorphism $M \hookrightarrow M' \upharpoonright_\varphi$ iff the expansion is **extending**,
- there is an arbitrary model homomorphism $M \to M' \upharpoonright_\varphi$ iff the expansion is **using**, and
- M' is the free over M with respect to φ (see Definition 19) iff the expansion is **free**.

$\Box$

The following extends the concept of liberality (free extensions) to extra theory morphisms:

[6] Under a suitable concept of "injectivity".

Definition 19 A extra theory morphism $\varphi\colon (\Sigma, E) \to (\Sigma', E')$ is **liberal** iff the reduct functor $\_{\restriction_\varphi}\colon \textsc{Mod}'(\Sigma', E') \to \textsc{Mod}(\Sigma, E)$ has a left adjoint, i.e., iff for any model $M \in |\textsc{Mod}(\Sigma, E)|$, there exists a model $M' \in |\textsc{Mod}'(\Sigma', E')|$ and a model morphism $\eta_M\colon M \to M'{\restriction_\varphi}$ such that for any model $N' \in |\textsc{Mod}'(\Sigma', E')|$ and any model morphism $h\colon M \to N'{\restriction_\varphi}$ there exists a unique model morphism $h'\colon M' \to N'$ such that $h = \eta_M ; h'{\restriction_\varphi}$.

$$\begin{tikzcd}
M \arrow[r, "\eta_M"] \arrow[dr, "h"'] & M'{\restriction_\varphi} \arrow[d, "h'{\restriction_\varphi}"] & M' \arrow[d, "h'"] \\
 & N'{\restriction_\varphi} & N'
\end{tikzcd}$$

□

The following results give sufficient conditions for "extra" liberality based on "intra" liberality.

Theorem 20 If $\mathfrak{I}$ is liberal on signature morphisms, $\mathfrak{I}'$ and (Φ, α, β) are liberal, then any extra theory morphism $\varphi\colon (\Sigma, E) \to (\Sigma', E')$ is liberal. Moreover, the free (Σ', E')-model over a given (Σ, E)-model M can be obtained as $\overline{\beta}_{\Sigma'}(M^\varphi)/E'$. □

Theorem 21 If (Φ, α, β) is a strong liberal embedding, $\mathfrak{I}'$ is liberal, and $\overline{\beta}$ satisfies the following Satisfaction Condition:

$$\overline{\beta}_{\Sigma'}(M) \models_{\Sigma'} \alpha_{\Sigma'}(e) \ \text{ if } \ M \models_{\Phi(\Sigma')} e$$

for all $M \in |\textsc{Mod}(\Phi(\Sigma')|$ and $e \in Sen(\Phi(\Sigma'))$, then each extra theory morphism $\varphi\colon (\Sigma, E) \to (\Sigma', E')$ is liberal. □

Corollary 22 If (Φ, α, β) is a persistent strong embedding and $\mathfrak{I}'$ is liberal, then any extra theory morphism is liberal. □

The co-limit of a diagram of extra theory morphisms is computed in a pre-defined fixed institution in which all institutions underlying the nodes of the diagram are embedded. This is more general than just doing it in the co-limit of the underlying diagram of institution embeddings; this generality is useful in applications since in the for multi-paradigm systems the co-cones of the underlying institutions are not necessarily co-limit co-cones. For details see [12], here we will mention only a corollary of the fundamental result on co-limits of theory morphisms:

Corollary 23 A diagram of extra theory morphisms has a co-limit whenever the co-limit co-cone of the underlying diagram of institution morphisms consists of institution embeddings. □

In general, extra theory morphisms do not necessarily support semi-exactness. Fortunately, some special cases of exactness for extra theory morphisms are enough to explain most practical situations. An important special case is given by the pushout between an intra and an extra theory morphism.

Theorem 24 Consider an institution embedding $(\Phi,\alpha,\beta)\colon \mathfrak{I} \to \mathfrak{I}_1$ and let $\varphi^{u2}\colon T \to T_2$ be a intra theory morphism in $\mathfrak{I}$, and $\varphi^{u1}\colon T \to T_1$ be a extra theory morphism with (Φ,α,β) the underlying institution morphism. If $\mathfrak{I}_1$ is (weakly) semi-exact and (Φ,α,β) is (weakly) additive and either of the following holds:

- (Φ,α,β) is strong, or
- $\mathfrak{I}$ is (weakly) semi-exact and Φ is surjective on objects and full

then the corresponding diagram of model reducts

$$\begin{tikzcd}
\mathrm{Mod}(T) & \mathrm{Mod}^1(T_1) \arrow[l, "-\restriction_{\varphi^{u1}}"'] \\
\mathrm{Mod}(T_2) \arrow[u, "-\restriction_{\varphi^{u2}}"] & \mathrm{Mod}^1(T_0) \arrow[l, "-\restriction_{\varphi^{2}}"] \arrow[u, "-\restriction_{\varphi^{1}}"']
\end{tikzcd}$$

is a (weak) pullback. □

Inclusion systems where first introduced by [14] for the institution-independent study of structuring specifications. They provide the underlying mathematical concept for module imports, which are the most fundamental structuring construct. In this paper we use the *weak inclusion systems* of [4], which constitute a improvement of the original definition of inclusion systems of [14].

Definition 25 $\langle \mathcal{I}, \mathcal{E}\rangle$ is a **weak inclusion system** for a category $\mathbb{C}$ if $\mathcal{I}$ and $\mathcal{E}$ are two sub-categories with $|\mathcal{I}| = |\mathcal{E}| = |\mathbb{C}|$ such that

1. $\mathcal{I}$ is a partial order, and
2. every arrow f in $\mathbb{C}$ can be factored uniquely as $f = e;i$ with $e \in \mathcal{E}$ and $i \in \mathcal{I}$.

The arrows of $\mathcal{I}$ are called **inclusions**, and the arrows of $\mathcal{E}$ are called **surjections**.[7] The domain (source) of the inclusion i in the factorization of f is called called the **image of** f and denoted as $\mathrm{Im}(f)$. An **injection** is a composite between an inclusion and an isomorphim. □

For the fundamental properties of weak inclusion systems and techniques to construct them consult [4].

We need the following technical definition:

Definition 26 Let $\mathbb{C}$ and $\mathbb{C}'$ be two categories with weak inclusion systems $\langle \mathcal{I}, \mathcal{E}\rangle$ and $\langle \mathcal{I}', \mathcal{E}'\rangle$ respectively. Then a functor $\mathcal{U}\colon \mathbb{C} \to \mathbb{C}'$ **lifts inclusions uniquely** iff for any inclusion $\iota'\colon A' \hookrightarrow B\mathcal{U}$ in $\mathcal{I}'$ with $B \in |\mathbb{C}|$, there exists a unique inclusion $\iota \in \mathcal{I}$ such that $\iota\mathcal{U} = \iota'$. □

Theorem 27 Consider a category of institutions with a weak inclusion system $\langle \mathcal{I}^{\mathrm{INST}}, \mathcal{E}^{\mathrm{INST}}\rangle$ such that each of institutions involved $\mathfrak{I} = (\mathbb{S}ign, \mathrm{Mod}, Sen, \models)$ has a weak inclusion system $\langle \mathcal{I}^{\mathfrak{I}}, \mathcal{E}^{\mathfrak{I}}\rangle$ for its category of signatures. If

[7] Surjections of some weak inclusion systems need not necessarily be surjective in the ordinary sense.

- Φ preserves inclusions for each $(\Phi,\alpha,\beta) \in I^{\mathrm{INST}}$, and
- Φ preserves both inclusions and surjections and lifts inclusions uniquely for each $(\Phi,\alpha,\beta) \in \mathcal{E}^{\mathrm{INST}}$,

then the corresponding category of extra theory morphisms has an inclusion system where $\varphi\colon (\Sigma,E) \to (\Sigma',E')$ is

- *inclusion* iff both the underlying institution morphism $(\Phi,\alpha,\beta)\colon \mathfrak{I} \to \mathfrak{I}'$ and the signature morphism $\varphi\colon \Sigma \to \Sigma'\Phi$ are inclusions,
- *surjection* iff both the underlying institution morphism $(\Phi,\alpha,\beta)\colon \mathfrak{I} \to \mathfrak{I}'$ and the signature morphism $\varphi\colon \Sigma \to \Sigma'\Phi$ are surjections, and if $\alpha_{\Sigma'}(\varphi(E))^{\bullet} = E'$.

□

Practical applications use mostly the following much simpler Corollary:

Corollary 28 Consider a partial ordered set of institutions and institution morphisms such that each of institutions involved $\mathfrak{I} = (\mathbb{S}ign, \mathrm{MOD}, Sen, \models)$ has a weak inclusion system $\langle I^{\mathfrak{I}}, \mathcal{E}^{\mathfrak{I}}\rangle$ for its category of signatures with Φ preserving inclusions for each institution morphism (Φ,α,β). Then the corresponding category of extra theory morphisms has an inclusion system where $\varphi\colon (\Sigma,E) \to (\Sigma',E')$ is

- *inclusion* iff the signature morphism $\varphi\colon \Sigma \to \Sigma'\Phi$ is an inclusion in $I^{\mathfrak{I}}$,
- *surjection* iff the underlying institution morphism is identity and it is a surjection in $\mathcal{E}^{\mathfrak{I}}$

□

B Hidden Order Sorted Rewriting Logic

We devote this appendix to the (rather informal) presentation in some detail of HOSRWL (first introduced in [11] in the many sorted version) which embeds all CafeOBJ cube institutions. However, the deep understanding of HOSRWL requires further reading on its main components ([33] for RWL and [20, 22] for HSA) as well as their integration [11]. We assume familiarity with basic many sorted algebra which constitutes the underlying level of all algebraic specification developments (relevant background can be found in [17, 27, 34]), but also with order sorted algebra [26, 19].

Signatures

A **hidden signature** is a tuple $(H,V,\leq,\Psi,\Sigma,\Sigma^{\mathrm{b}})$, where

- $(H,\leq)$ is a partially ordered set of **hidden sorts**,
- $(V,\leq)$ is a partially ordered set of **visible sorts**,
- $(H,\leq)$ and $(V,\leq)$ are disjoint,
- Ψ is an $(V,\leq)$-order-sorted (o.s., for short) signature,
- Σ is an $(H \cup V,\leq)$-o.s. signature,

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

- $\Sigma^{\mathrm{b}} \subseteq \Sigma$ is a marked sub-signature of **behavioural operations** such that $\Sigma^{\mathrm{b}} \cap \Psi = \emptyset$, and

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

The operations in Σ^{b} have object-oriented meaning, $\sigma \in \Sigma^{\mathrm{b}}_{w,s}$ is **method** if s is hidden and **attribute** if s is visible. 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 $(H, V, \leq, \Psi, \Sigma, \Sigma^{\mathrm{b}}, E)$ where $(H, V, \leq, \Psi, \Sigma, \Sigma^{\mathrm{b}})$ is a hidden o.s. signature and E is a collection of Σ-equations. A **hidden sorted rewrite signature morphism** $\phi\colon (H, V, \leq, \Psi, \Sigma, \Sigma^{\mathrm{b}}, E) \to (H', V', \leq, \Psi', \Sigma', \Sigma' b, E')$ is an o.s. signature morphism $(H \cup V, \leq, \Sigma) \to (H' \cup V', \leq, \Sigma')$ such that

(M1) $\phi(\Psi) \subseteq \Psi'$,

(M2) $\phi(H) \subseteq H'$ and $\phi(\Sigma^{\mathrm{b}}) \subseteq \Sigma' b$,

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

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

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

The first two conditions say that hidden sorted signature morphisms preserve visibility and invisibility for both sorts and operations, the third[8] and fourth conditions express the encapsulation of classes and subclasses (in the sense that no new methods or attributes can be defined on an imported class), while the fifth expresses the encapsulation 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) **transition** (modulo E). Since equations are very traditional to algebraic specification, we concentrate here on transitions. A conditional transition is written as

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

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

Given a signature morphism $\phi\colon (H, V, \leq, \Psi, \Sigma, \Sigma^{\mathrm{b}}, E) \to (H', V', \leq, \Psi', \Sigma', \Sigma' b, E')$ the translation of sentences is defined by the translation of Σ-terms (modulo E) to Σ'-terms modulo E' along ϕ by replacing all symbols in Σ-terms with the corresponding symbols for Σ'. Condition (M5) enforces the correctness of this definition. For a full rigorous treatment of this issue the reader is advised to consult [5, 9].

Models

Given an algebraic theory (Σ, E), a **rewrite model** for (Σ, 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, each subsort relation $s < s'$ as sub-category relation $M_s \subseteq M_{s'}$, and each operation $\sigma \in \Sigma_{w,s}$ as a functor $\sigma_M\colon M_w \to M_s$, where M_w stands for $M_{s_1} \times \ldots \times M_{s_n}$ for $w = s_1 \ldots s_n$. Each Σ-term $t\colon w \to s$ gets a functor $t_M\colon M_w \to M_s$ by evaluating it for each assignment of the variables occurring in t with arrows from the corresponding carriers of M.

[8]Without (M3) the Satisfaction Condition fails, for more details on the logical and computational relevance of (M3) see [20].

The satisfaction of an equation $t = t'$ by M is given by $t_M = t'_M$;[9] in particular all structural equations should be satisfied by M. A model morphism is a family of functors indexed by the sorts commuting with the interpretations of the operations in Σ.

This algebra "enriched" over $\mathbb{C}at$ is a special case of *category-based equational logic* (see [5, 6, 21]) when letting the category $\mathbb{A}$ of models be the interpretations of Σ 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 $\mathcal{U}\colon \mathbb{A} \to \mathbb{X}$ 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 aid to the model theory of RWL.

Hidden sorted models are just ordinary models (either algebras or rewrite models).

Satisfaction

Let $(H, \leq, \Sigma, E)$ be a hidden sorted signature, $[\rho]$ be a sentence,[10] and M be a model for this signature. Satisfaction in RWL of ordinary equations was explained in the paragraph on models, so we concentrate on the satisfaction of transitions.

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

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

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

Finally, the satisfaction in HOSRWL is **behavioural** (denoted by $\models\!\!\equiv$); for details see [11, 7].

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

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