# The Common Framework Initiative for algebraic specification and development of software[⋆]

Donald Sannella

Laboratory for Foundations of Computer Science
University of Edinburgh, UK

dts@dcs.ed.ac.uk    www.dcs.ed.ac.uk/~dts/

**Abstract.** The Common Framework Initiative (CoFI) is an open international collaboration which aims to provide a common framework for algebraic specification and development of software. The central element of the Common Framework is a specification language called Casl for formal specification of functional requirements and modular software design which subsumes many previous algebraic specification languages. This paper is a brief summary of past and present work on CoFI.

## 1 Introduction

*Algebraic specification* is one of the most extensively-developed approaches in the formal methods area. The most fundamental assumption underlying algebraic specification is that programs are modelled as *many-sorted algebras* consisting of a collection of sets of data values together with functions over those sets. This level of abstraction is commensurate with the view that the correctness of the input/output behaviour of a program takes precedence over all its other properties. Another common element is that specifications of programs consist mainly of logical *axioms*, usually in a logical system in which equality has a prominent role, describing the properties that the functions are required to satisfy. This *property-oriented* approach is in contrast to so-called *model-oriented* specifications in frameworks like VDM which consist of a simple realization of the required behaviour. Confusingly — because the theoretical basis of algebraic specification is largely in terms of constructions on algebraic models — it is at the same time much more model-oriented than approaches such as those based on type theory (see e.g. [NPS90]), where the emphasis is almost entirely on syntax and formal systems of rules while semantic models are absent or regarded as of secondary importance.

The past 25 years has seen a great deal of research on the theory and practice of algebraic specification. Overviews of this material include [Wir90], [BKLOS91], [LEW96], [ST97], [AKK99] and [ST??]. Developments on the foundational side have been balanced by work on applications, but despite a number of success stories, industrial adoption has so far been limited. The proliferation of

[⋆] This research was supported by the ESPRIT-funded CoFI Working Group.

*algebraic specification languages* is seen as a significant obstacle to the dissemination and use of these techniques. Despite extensive past collaboration between the main research groups involved and a high degree of agreement concerning the basic concepts, the field has given the appearance of being extremely fragmented, with no *de facto* standard specification language, let alone an international standard. Moreover, although many tools supporting the use of algebraic techniques have been developed in the academic community, none of them has gained wide acceptance, at least partly because of their isolated usability: each tool uses a different specification language.

Since late 1995, work has been underway in an attempt to remedy this situation. The *Common Framework Initiative* (abbreviated CoFI) is an open international collaboration which aims to provide a common framework for algebraic specification and development of software. The Common Framework is intended to be attractive to researchers in the field as a common basis for their work, and to ultimately become attractive for use in industry. The central element of the Common Framework is a specification language called Casl (the Common Algebraic Specification Language), intended for formal specification of functional requirements and modular software design and subsuming many previous specification languages. Development of prototyping and verification tools for Casl will lead to them being interoperable, i.e. capable of being used in combination rather than in isolation.

Most effort to date has concentrated on the design of Casl, which concluded in late 1998. Even though the intention was to base the design on a critical selection of concepts and constructs from existing specification languages, it was not easy to reach a consensus on a coherent language design. A great deal of careful consideration was given to the effect that the constructs available in the language would have on such aspects as the methodology for formal development of modular software from specifications and the ease of constructing appropriate support tools. A complete formal semantics for Casl was produced in parallel with the later stages of the language design, and the desire for a relatively straightforward semantics was one factor in the choice between various alternatives in the design. Work on CoFI has been an activity of Ifip WG 1.3 and the design of Casl has been approved by this group.

This paper is a brief summary of work in CoFI with pointers to information available elsewhere. Casl is given special prominence since it is the main concrete product of CoFI so far. A more extensive description of the rationale behind CoFI and Casl may be found in [Mos97] and [Mos99].

## 2 CASL

Casl represents a consolidation of past work on the design of algebraic specification languages. With a few minor exceptions, all its features are present in some form in other languages but there is no language that comes close to subsuming it. Designing a language with this particular novel collection of features required solutions to a number of subtle problems in the interaction between features.

It soon became clear that no single language could suit all purposes. On one hand, sophisticated features are required to deal with specific programming paradigms and special applications. On the other, important methods for prototyping and reasoning about specifications only work in the *absence* of certain features: for instance, term rewriting requires specifications with equational or conditional equational axioms.

CASL is therefore the heart of a *family* of languages. Some tools will make use of well-delineated *sub-languages* of CASL obtained by syntactic or semantic restrictions, while *extensions* of CASL will be defined to support various paradigms and applications. The design of CASL took account of some of the planned extensions, particularly one that involves higher-order functions [MHK98], and this had an important impact on decisions concerning matters like concrete syntax.

CASL consists of the following major parts or "layers": basic specifications; structured specifications; architectural specifications; specification libraries. A detailed description of the features of CASL may be found in [Mos99] and the complete language definition is in [CoFI98]. Here we just give a quick overview and a couple of simple examples in the hope that this will give a feeling for what CASL is like. Further examples may be found in the appendices of [CoFI98]. Since features of various existing specification languages have found their way into CASL in some form, there are of course many interesting relationships with other languages. It is not the purpose of this paper to detail these so many relevant references are omitted.

A CASL basic specification denotes a class of *many-sorted partial first-order structures*: algebras where the functions are partial or total, and where also predicates are allowed. These are classified by *signatures*, which list sort names, partial and total function names, and predicate names, together with profiles of functions and predicates. The sorts are partially ordered by a subsort inclusion relation, which is interpreted as embedding rather than set-theoretic inclusion, and is required to commute with overloaded functions. A CASL basic specification includes *declarations* to introduce components of signatures and *axioms* to give properties of structures that are to be considered as *models* of a specification. Axioms are written in first-order logic (so, with quantifiers and the usual logical connectives) built over atomic formulae which include strong and existential equalities, definedness formulae and predicate applications, with generation constraints added as special, non-first-order sentences. The interpretation of formulae is as in classical two-valued first-order logic, in contrast to some frameworks that accommodate partial functions. Concise syntax is provided for specifications of "datatypes" with constructor and selector functions.

Here is an example of a basic specification:

**free types** $\mathit{Nat} ::= \mathit{0} \mid$ **sort** $\mathit{Pos}$;
$\qquad\qquad\mathit{Pos} ::= \mathit{suc}(\mathit{pre} : \mathit{Nat})$
**op** $\mathit{pre} : \mathit{Nat} \rightarrow ?\ \mathit{Nat}$
**axioms**
$\qquad \neg \mathit{def}\ \mathit{pre}(\mathit{0})$;
$\qquad \forall n : \mathit{Nat} \bullet \mathit{pre}(\mathit{suc}(n)) = n$

```
pred  $even\_\_ : Nat$
var   $n : Nat$
•     $even\ 0$
•     $even\ suc(n) \Leftrightarrow \neg even\ n$
```

The remaining features of CASL do not depend on the details of the features for basic specifications, so this part of the design is orthogonal to the rest. An important consequence of this is that sub-languages and extensions of CASL can be defined by restricting or extending the language of basic specifications (under certain conditions) without the need to reconsider or change the rest of the language.

CASL provides ways of building complex specifications out of simpler ones (the simplest ones being basic specifications) by means of various *specification-building operations*. These include translation, hiding, union, and both free and loose forms of extension. A structured specification denotes a class of many-sorted partial first-order structures, as with basic specifications. Thus the structure of a specification is *not* reflected in its models: it is used only to present the specification in a modular style. Structured specifications may be named and a named specification may be *generic*, meaning that it declares some *parameters* that need to be *instantiated* when it is used. Instantiation is a matter of providing an appropriate *argument specification* together with a *fitting morphism* from the parameter to the argument specification. Fitting may also be accomplished by the use of named *views* between specifications. Generic specifications correspond to what is known in other specification languages as (*pushout-style*) *parametrized specifications.*

Here is an example of a generic specification (referencing a specification named PARTIAL_ORDER, which is assumed to declare the sort *Elem* and the predicate $\_\_ \leq \_\_$):

```
spec LIST_WITH_ORDER [PARTIAL_ORDER] =
  free type $List[Elem] ::= nil \mid cons(hd :?Elem; tl :?List[Elem])$
then
  local
      op  $insert : Elem \times List[Elem] \to List[Elem]$;
      vars  $x, y : Elem; l : List[Elem]$
      axioms  $insert(x, nil) = cons(x, nil)$;
              $x \leq y \Rightarrow insert(x, cons(y, l)) = cons(x, insert(y, l))$;
              $\neg(x \leq y) \Rightarrow insert(x, cons(y, l)) = cons(y, insert(x, l))$
  within
      pred  $order[\_\_ \leq \_\_] : List[Elem] \times List[Elem]$
      vars  $x : Elem; l : List[Elem]$
      axioms  $order[\_\_ \leq \_\_](nil) = nil$;
              $order[\_\_ \leq \_\_](cons(x, l)) = insert(x, order[\_\_ \leq \_\_](l))$
end
```

Architectural specifications in CASL are for describing the modular structure of software, in constrast to structured specifications where the structure is only for presentation purposes. Architectural specifications are probably the most novel aspect of CASL; they are not entirely new, but they have no counterpart in most algebraic specification languages. An architectural specification consists of a list of *unit declarations*, indicating the component modules required with specifications for each of them, together with a *unit term* that describes the way in which these modules are to be combined. (There is an unfortunate potential for confusion here: in CASL, the term "architecture" refers to the "implementation" modular structure of the system rather than to the "interaction" relationships between modules in the sense of [AG97].) Units are normally functions which map structures to structures, where the specification of the unit specifies properties that the argument structure is required to satisfy as well as properties that are guaranteed of the result. These functions are required to be *persistent*, meaning that the argument structure is preserved intact in the result structure. This corresponds to the fact that a software module must use its imports as supplied without altering them.

Here is a simple example of an architectural specification (referencing ordinary specifications named LIST, CHAR, and NAT, assumed to declare the sorts *Elem* and *List*[*Elem*], *Char*, and *Nat*, respectively):

**arch spec** CN_LIST =
  **units**
    $C$ : CHAR ;
    $N$ : NAT ;
    $F$ : ELEM $\rightarrow$ LIST[ELEM]
  **result** $F[C$ **fit** $\mathit{Elem} \mapsto \mathit{Char}]$ **and** $F[N$ **fit** $\mathit{Elem} \mapsto \mathit{Nat}]$

More about architectural specifications, including further examples, may be found in [BST99].

Libraries in CASL are collections of named specifications. A specification can refer to an item in a library by giving its name and the location of the library that contains it. CASL includes direct support for establishing distributed libraries on the Internet with version control.

## 3 Semantics

The formal semantics of CASL, which is complete but whose presentation still requires some work, is in [CoFI99]. The semantics is divided into the same parts as the language definition (basic specifications, structured specifications, etc.) but in each part there is also a split into *static semantics* and *model semantics*.

The static semantics checks well-formedness of phrases and produces a "syntactic" object as result, failing to produce any result for ill-formed phrases. For example, for a basic specification the static semantics yields a *theory presentation* containing the sorts, function symbols, predicate symbols and axioms that belong to the specification. (Actually it yields an *enrichment*: when a basic specification is used to extend an existing specification it may refer to existing sorts, functions and predicates.) A phrase may be ill-formed because it makes reference to non-existent identifiers or because it contains a sub-phrase that fails to type check. The *model semantics* provides the corresponding model-theoretic part of the semantics, and is intended to be applied only to phrases that are well-formed according to the static semantics. For a basic specification, the model semantics yields a class of models. A statically well-formed phrase may still be ill-formed according to the model semantics: for example, if a generic specification is instantiated with an argument specification that has an appropriate signature but which has models that fail to satisfy the axioms in the parameter specification, then the result is undefined. The judgements of the static and model semantics are defined inductively by means of rules in the style of Natural Semantics.

The orthogonality of basic specifications in Casl with respect to the rest of the language is reflected in the semantics by the use of a variant of the notion of institution [GB92] called an *institution with symbols* [Mos98]. (For readers who are unfamiliar with the notion of institution, it corresponds roughly to "logical system appropriate for writing specifications".) The semantics of basic specifications is regarded as defining a particular institution with symbols, and the rest of the semantics is based on an arbitrary institution with symbols.

The semantics provides a basis for the development of a proof system for Casl. As usual, at least three levels are needed: proving consequences of sets of axioms; proving consequences of structured specifications; and finally, proving the refinement relation between structured specifications. The semantics of Casl gives a reference point for checking the soundness of each of the proposed proof systems and for studying their completeness.

## 4 Methodology

The original motivation for work on algebraic specification was to enable the stepwise development of correct software systems from specifications with verified refinement steps. Casl provides good support for the production of specifications both of the problem to be solved and of components of the solution, but it does not incorporate a specific notion of refinement. Architectural specifications go some way towards relating different stages of development but they do not provide the full answer. Other methodological issues concern the "endpoints" of the software development process: how the original specification is obtained in the first place (requirements engineering), and how the transition is made from Casl to a given programming language. Finally, the usual issues in programming methodology are relevant here, for instance: verification versus testing; software reuse and specification reuse; software reverse engineering; software evolution.

Casl has been designed to accommodate multiple methodologies. Various existing methodologies and styles of use of algebraic specifications have been considered during the design of Casl to avoid unnecessary difficulties for users who are accustomed to a certain way of doing things. For the sake of concreteness, the present author prefers the methodology espoused in [ST97], and work on adapting this methodology to Casl has begun.

## 5 Support tools

Tool activity initially focussed on the concrete syntax of Casl to provide feedback to the language design since the exact details of the concrete syntax can have major repercussions for parsing. Casl offers a flexible syntax with *mixfix* notation for application of functions and predicates to arguments, which requires relatively advanced parsing methods. ASF+SDF was used to prototype the Casl syntax in the course of its design, and several other parsers have been developed concurrently. Also available is a LaTeX package for uniform formatting of Casl specifications with easy conversion to HTML format. ATerms [BKO98] have been chosen as the common interchange format for CoFI tools. This provides a tree representation for various objects (programs, specifications, abstract syntax trees, proofs) and annotations to store computed results so that one tool can conveniently pass information to another. Work is underway on a format for annotations and on a list of specific kinds of annotations.

At present, the principal focus of tools work in CoFI is on adapting tools that already exist for use with Casl. Existing rewrite engines such as in OBJ, ASF+SDF and ELAN should provide a good basis for prototyping (parts of) Casl specifications. For verification tools, we plan to reuse existing proof tools for specific subsets of Casl: equational, conditional, full first-order logic with total functions, total functions with subsorts, partial functions, etc. The integration of proof tools such as SPIKE, EXPANDER and others will provide the potential to perform proofs by induction, observational proofs, termination proofs, etc. One system on which development is already well-advanced is HOL-CASL [MKK98] which provides static analysis of Casl specifications and theorem proving via an encoding into the Isabelle/HOL theorem prover [Pau94]. Another is INKA 5.0 [AHMS99] which provides theorem proving for a sub-language of Casl that excludes partial functions.

## 6 Specification of reactive systems

An area of particular interest for applications is that of reactive, concurrent, distributed and real-time systems. There is considerable past work in algebraic specification that tackles systems of this kind, but nonetheless the application of Casl to such systems in speculative and preliminary in comparison with the rest of CoFI. The aim here is to propose and develop one or more extensions of CASL to deal with systems of this kind, and to study methods for developing software from such specifications. Extensions in three main categories are currently being considered:

– Combination of formalisms for concurrency (e.g. CCS, Petri nets, CSP) with Casl for handling classical (static) data structures;
– Formalisms built over Casl, where processes are treated as special dynamic data; and
– Approaches where CASL is used for coding at the meta-level some formalism for concurrency, as an aid to reasoning.

Work in this area begun only after the design of CASL was complete and so it is still in its early stages.

## 7 Invitation

COFI is an open collaboration, and new participants are welcome to join at any time. Anybody who wishes to contribute is warmly invited to visit the COFI web site at `http://www.brics.dk/Projects/CoFI/` where all COFI documentation, design notes, minutes of past meetings etc. are freely available. Announcements of general interest to COFI participants are broadcast on the low-volume mailing list `cofi-list@brics.dk` and each task group has its own mailing list; see the COFI web site for subscription instructions. All of these mailing lists are moderated. Funding from the European Commission is available until September 2000 to cover travel to COFI meetings although there are strict rules concerning eligibility, see `http://www.dcs.ed.ac.uk/home/dts/CoFI-WG/`.

**Acknowledgements** Many thanks to all the participants of COFI, and in particular to the coordinators of the various COFI Task Groups: Bernd Krieg-Brückner (Language Design); Andrzej Tarlecki (Semantics); Michel Bidoit (Methodology); Hélène Kirchner (Tools); Egidio Astesiano (Reactive Systems); and especially Peter Mosses (External Relations) who started COFI and acted as overall coordinator until mid-1998.

## References

[AG97] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, July 1997.

[AKK99] E. Astesiano, H.-J. Kreowski and B. Krieg-Brückner (eds.). *Algebraic Foundations of Systems Specification.* Springer (1999).

[AHMS99] S. Autexier, D. Hutter, H. Mantel and A. Schairer. Inka 5.0: a logic voyager. *Proc. 16th Intl. Conference on Automated Deduction*, Trento. Springer LNAI 1632, 207–211 (1999).

[BKLOS91] M. Bidoit, H.-J. Kreowski, P. Lescanne, F. Orejas and D. Sannella (eds.). *Algebraic System Specification and Development: A Survey and Annotated Bibliography.* Springer LNCS 501 (1991).

[BST99] M. Bidoit, D. Sannella and A. Tarlecki. Architectural specifications in CASL. *Proc. 7th Intl. Conference on Algebraic Methodology and Software Technology*, Manaus, Brazil. Springer LNCS 1548, 341–357 (1999).

[BKO98] M. van den Brand, P. Klint and P. Olivier. ATerms: exchanging data between heterogeneous tools for CASL. COFI Note T-3, `http://www.brics.dk/Projects/CoFI/Notes/T-3/` (1998).

[CoFI98] COFI Task Group on Language Design. CASL – The COFI algebraic specification language – Summary (version 1.0). `http://www.brics.dk/Projects/CoFI/Documents/CASL/Summary/` (1998).

[CoFI99] COFI Task Group on Semantics. CASL – The COFI algebraic specification language – Semantics (version 1.0). COFI Note S-9, `http://www.brics.dk/Projects/CoFI/Notes/S-9/` (1999).

[GB92] J. Goguen and R. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the Assoc. for Computing Machinery* 39:95–146 (1992).

[Mos98] T. Mossakowski. Institution-independent semantics for Casl-in-the-large. CoFI Note S-8, `http://www.brics.dk/Projects/CoFI/Notes/S-8/` (1998).

[MHK98] T. Mossakowski, A. Haxthausen and B. Krieg-Brückner. Subsorted partial higher-order logic as an extension of Casl. CoFI Note L-10, `http://www.brics.dk/Projects/CoFI/Notes/L-10/` (1998).

[MKK98] T. Mossakowski, Kolyang and B. Krieg-Brückner. Static semantic analysis and theorem proving for Casl. *Recent Trends in Algebraic Development Techniques: Selected Papers from WADT'97*, Tarquinia. Springer LNCS 1376, 333–348 (1998).

[LEW96] J. Loeckx, H.-D. Ehrich and M. Wolf. *Specification of Abstract Data Types.* Wiley (1996).

[Mos97] P. Mosses. CoFI: the common framework initiative for algebraic specification and development. *Proc. 7th Intl. Joint Conf. on Theory and Practice of Software Development*, Lille. Springer LNCS 1214, 115–137 (1997).

[Mos99] P. Mosses. Casl: a guided tour of its design. *Recent Trends in Algebraic Development Techniques: Selected Papers from WADT'98*, Lisbon. Springer LNCS 1589, 216–240 (1999).

[NPS90] B. Nordström, K. Petersson and J. Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford Univ. Press (1990).

[Pau94] L. Paulson. *Isabelle: A Generic Theorem Prover.* Springer LNCS 828 (1994).

[ST97] D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing* 9:229–269 (1997).

[ST??] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specifications and Formal Program Development.* Cambridge Univ. Press, to appear.

[Wir90] M. Wirsing. Algebraic specification. *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.). North-Holland (1990).