# 10

# Foundations

**Donald Sannella and Andrzej Tarlecki**

> *A complete presentation of* CASL *is in the* Reference Manual.

This *User Manual* has introduced the potential user to the features of CASL mainly by means of illustrative examples. It has presented and discussed the typical ways in which the language concepts and constructs are expected to be used in the course of building system specifications. Thus, the presentation in Part II focused on *what* the constructs and concepts of CASL are *for*, and *how* they should (and should not) be used. We tried to make these points as clear as possible by referring to simple examples, and by discussing both the general ideas and some details of CASL specifications. We hope that this has given the reader a sufficient feel of the formalism, and enough understanding, to look through the presentation of a basic library of CASL specifications in Chap. 12, to follow the case study in Chap. 13, and to start experimenting with the use of CASL for writing specifications – perhaps employing the support tools presented in Chap. 11.

By no means, however, should this book be regarded as a complete presentation of the CASL specification formalism – this is given in the accompanying volume, the CASL *Reference Manual* [20].

> *CASL has a definitive summary.*

The CASL Reference Manual begins with a definitive *summary* of the entire CASL language: all the language constructs are listed there systematically, together with the syntax used to write them down and a detailed explanation of their intended meaning. However, although it tries to be precise and complete, the CASL Summary still relies on natural language to present CASL.

This inherently leaves some room for interpretation and ambiguity in various corners of the language, for example where details of different constructs interact.

*Casl has a complete formal definition.*

A key aim of the entire CoFI initiative is to avoid any potential ambiguities by providing a complete formal definition for Casl, and sound mathematical foundations for the advocated methodology of its use in software specification and development.

*Abstract and concrete syntax of Casl are defined formally.*

The next part of the Reference Manual is a formal definition of the syntax of Casl. *Abstract syntax* is given, where each phrase is written in a way that directly indicates its components, thus making evident its internal structure. In essence, the use of each construct of the language is explicitly labeled here. This is convenient for formal manipulation and analysis, but is not so readable. Therefore, the so-called *concrete syntax* of Casl (as used for instance in the examples throughout this book) is given as well, retaining a direct correspondence with the abstract syntax. This offers to the user of Casl a convenient and readable way of writing down Casl specifications, in a way that makes clear the formal structure of the phrases and constructs used to build them. As usual, the syntax is given as a context-free grammar, using a variant of the BNF notation, relying on well-established theory to give its formal meaning, and on a variety of tools and techniques available for syntactic analysis of languages presented in such a style.

*Casl has a complete formal semantics.*

The ultimate definition of the meaning of Casl specifications is provided by the *semantics* of Casl in the Reference Manual. The semantics first defines mathematical entities that formally model the intended meaning of various concepts underlying Casl, as already hinted at in Chap. 2, and further introduced and discussed throughout the summary.

The key concepts here are that of Casl signature, model and formula, together with the satisfaction relation between models and formulas over a common signature. In fact, these are variants of the standard algebraic and logical notions, thus linking work on Casl to well-established mathematical theories and ideas.

In a more or less standard way, we use these concepts to build the *semantic domains* which the meanings of phrases in various syntactic categories of CASL inhabit. We have chosen to give the semantics in the form of so-called *natural semantics*, with formal deduction rules to derive judgments concerning the meaning of each CASL phrase from the meanings of its constituent parts. Not only is this a mathematically well-established formalism with an unambiguous interpretation, but we also hope that this makes the semantics itself more readable, with details easier to follow than in some other approaches.

The overall semantics of CASL consists of two parts. The *static semantics* captures a form of static analysis of CASL specifications, in which they are checked for well-formedness – for example, that axioms are well-typed, and references to named entities are in scope. Then the *model semantics* takes a well-formed CASL specification and assigns a model-theoretic meaning to it.

*CASL specifications denote classes of models.*

In CASL, well-formed specifications denote signatures (static semantics) and classes of models (model semantics). Basic specifications, which in essence present a signature and a set of axioms over this signature, denote the class of models that satisfy all the axioms. The semantics of basic specifications is split into two parts: first many-sorted basic specifications are described and then features for defining and using subsorts are added.

*The semantics is largely institution-independent.*

A few more concepts are needed to explain the semantics of structured specifications. Key here is the notion of signature morphism, and the model reducts and translation of sentences that signature morphisms induce. Having introduced those, we obtain the *institution* [23] of CASL, that is, the underlying logical system of CASL equipped with extra structure to capture ways of moving between signatures linked by signature morphisms. One important point of the semantics is that all the layers of the semantics "above" basic specifications are *institution-independent*, i.e., well-defined for any institution chosen to build basic specifications (as long as the institution comes with a bit of extra structure concerned with forming unions of signatures and defining signature morphisms).

Next, we have the semantics of architectural specifications, which relies on a formal counterpart of the concept of a unit (module) of a system to be developed: self-contained units are viewed simply as models of the underlying institution, and parametrized units as functions mapping such parameter models to result models. Architectural specifications provide a way of specifying the component units of a system and indicating how the overall system is built by putting these units together. This intuition is captured by the semantics of architectural specifications, which denote a class of permitted unit bindings and a function that maps each such environment to a result unit.

Finally, the libraries layer of Casl is given a rather standard description with libraries modeled as environments giving names to the entities introduced (specifications, architectural specifications, etc.).

*The semantics is the ultimate reference for the meanings of all Casl constructs.*

Overall, the formal mathematical semantics is crucial in the understanding of Casl specifications. It provides their unambiguous meaning, and thus gives the ultimate reference point for all questions concerning the interpretation of any Casl phrase in any context.

We have already experienced how important such a formal semantics may be in the design of Casl itself. In many cases, the intended semantics was prominent in internal discussions on the details of the constructs under consideration, and provided guidelines for many choices in the design of Casl. Indeed, the concrete syntax of Casl was designed only after the semantics was settled.

*Proof systems for various layers of Casl are provided.*

The semantics is also a necessary prerequisite for the development of mechanisms for formal reasoning about Casl specifications. This is the role of *proof calculi* that support reasoning about the various layers of Casl. The starting point is a formal system of deduction rules which determines a proof-theoretic counterpart of the consequence relation between sets of formulas, thus providing a way for deriving consequences of sets of axioms in Casl specifications. This is then extended to systems of rules for deriving consequences of structured specifications and for proving inclusions between classes of models of such specifications. These systems are also used in rules for formal verification of the internal correctness of system designs as captured by architectural specifications. For all these systems, their soundness is proved and completeness discussed by reference to the formal semantics of Casl.

One point of interest is that, again, the extension of the basic proof system for consequences between sets of formulas to structured and architectural specifications does not really rely on the specifics of the underlying institution, but just reflects the way in which the structuring and architectural constructs are defined for an arbitrary institution.

*The foundations of our CASL are rock-solid!*

All this work on the mathematical underpinnings of CASL and related specification and development methodology, as documented in the Reference Manual, should make it exceptionally trustworthy – at least in the sense that it provides a formal point of reference against which all the claims may (and should) be checked.