Clear: Language Design
Version 0.1

Don Sannella
The Common Framework Initiative

January 10, 1996

This document is part of the CFI Catalogue of Algebraic Specification Frameworks[1]. It was provided by *Don Sannella*; any comments and corrections should be addressed to `dts@dcs.ed.ac.uk`.

It will be made available formatted for printing as compressed Postscript[2] (~350Kb) and DVI[3] (~50Kb).

The WWW version provides only a rough approximation (mostly generated automatically by Hyperlatex) to the symbols used in the formatted versions. Please inform `pdmosses@brics.dk` about any places where an alternative approximation is desirable!

N.B. All sections of the catalogue are *TENTATIVE* first versions, unless explicitly marked otherwise. Sections dealing with Methodology, Tools, and Documentation will be added later.

1 Specification in the Small

1.1 Basic Constructs

1.1.1 signatures

Characterize:

Signatures are boring old algebraic signatures. (But for this and other features for specification in the small, see 2.1.2 below. In some papers, error handling à la Goguen(1977) is used.)

Notation for declaring sorts/types:

sorts *s1*,...,*sn*

Notation for declaring operations/functions, predicates/relations:

Operator declarations have the form **opns** *od1* ... *odn* where each *od* has the form $o : s1, \ldots, sn \rightarrow s$ for $n \geq 0$. $o : s$ is an abbreviation for $o :\rightarrow s$ and $o1, \ldots, om : s1 \ldots, sn \rightarrow s$ is an abbreviation for a set of m declarations. There are no predicates.

Notation for declaring variables:

$x1, \ldots, xn : s$ (variables are declared separately within each axiom)

Any restrictions on order of declarations:

Sorts first, then operations.

Notation for indicating local/private/hidden items:

Hiding is done using the **derive** operation (see 2.2.4 below).

[1] http://www.brics.dk/Projects/CFI/Catalogue
[2] ftp://ftp.brics.dk/pub/BRICS/Projects/CFI/Catalogue/Clear.ps.Z
[3] ftp://ftp.brics.dk/pub/BRICS/Projects/CFI/Catalogue/Clear.dvi

1.1.2 axioms

Characterize:
Universally quantified equations.
Notation:
all $X.t1 = t2$

1.1.3 models

Characterize the structure of a model:
Boring old total algebras.
Do specifications have loose semantics?
Yes.

1.1.4 constraints

Characterize (initiality, (free) generating, etc.):
Data constraints (i.e. free generating constraints).
Notation:
enrich ... **by data** ...

1.1.5 status

Can one indicate the status of items and axioms, e.g., differentiate between constructors/selectors, axioms/theorems, equations/rewrite-rules?
No.
Notation:

1.1.6 other

Any feature not covered above:
Any sort s specified as **data** comes equipped with an equality operation $==: s, s \rightarrow bool$

1.2 Types

Is type-checking decidable?
Yes.

1.2.1 predefined types

Notation for predefined types (e.g., Booleans):
bool

1.2.2 algebra of types

Notation for predefined type constructors (e.g., enumerations, records):
None available.

1.2.3 parametrized type constructors

Notation for declaring parametrized type constructors:
None available.

1.2.4 polymorphic types

How is polymorphism expressed: free variables, quantification, subtypes?
Not available.
Notation:
Can the effect of type classes be achieved?
The requirement of a parameterised specification may be something like PO; otherwise, no.
Notation:

1.2.5 dependent types

Can types depend on values (e.g., lists of definite length)?
No.
Notation:
If not, can the same effect be achieved by other means?
Only by the kind of clumsy encoding that is possible in any specification language.

1.2.6 other

Any feature not covered above:

1.3 Partiality

Can partiality be represented?
No.

1.3.1 subtypes defined by predicates

Are such 'semantic' subtypes allowed?
No.
Notation:
Is 'subtype checking' decidable?

1.3.2 subtypes declared in the signature

Notation:
Not available.
Is the interpretation of functions 'universal' or 'overloaded'?
Are 'retracts' available?

1.3.3 equations

Notation:
all $X.t1 = t2$
Interpretation (existential, strong, weak):
As usual in total algebras.

1.3.4 (un)definedness

Notation for asserting definedness:
Not available.
Notation for asserting undefinedness:
Canonical undefined term:

1.3.5 logic

Notation and interpretation for logical connectives:
None built in; of course, they can be encoded as usual.
If the logic is two-valued: can truth-values be undetermined/irrelevant?
No, except insofar as loose specifications allow one to regard them as such.

1.3.6 strictness

Are all operations strict w.r.t. undefinedness?
If so, how is the effect of an if-then-else operation provided?
Can exception-handling and/or error-recovery be expressed?

1.3.7 other

Any feature not covered above:

1.4 Notations

1.4.1 offside rule

Is layout significant for parsing?
No.
If so, explain how, e.g., 'offside rule':

1.4.2 graphical symbols

Are non-ASCII characters allowed?
Yes. (But lexical aspects of the language were never defined carefully.)
If so, is there always an equivalent ASCII representation?
No.

1.4.3 overloading

Can the same declared operation symbol be used on distinct (unrelated) types?
Yes.
Are there any built-in overloaded operation symbols?
== (see 1.1.6).

1.4.4 mixfix notation

Can applications of declared operations/predicates be written prefix, infix, postfix, mixfix?
Yes. (But such aspects of the language were never defined carefully.)
Notation for declaring fixity:
Example: $\_ + \_ : \mathit{int},\mathit{int} \rightarrow \mathit{int}$ (in some papers).
Notation for applications with alternative fixity, if any:
Can operations with varying numbers of arguments be declared?
No.
Notation:

1.4.5 operator precedence

Can grouping in terms be disambiguated using precedence?
No.
If so, how is precedence declared/determined?

1.4.6 associativity

Can (binary) operations be declared associative?
No.
Can (binary) operations be declared left- or right-associative?
Notation:

1.4.7 values of predefined types

Notation:
true, false

1.4.8 other

Any feature not covered above:

1.5 Abbreviations

1.5.1 type inference

Can instantiation of polymorphic operations be inferred? If so: always, usually, or only sometimes?
No polymorphism available.
Are explicit 'casts' allowed?
Notation:

1.5.2 conditionals

Can several conditional axioms with the same tests be combined?
No conditional axioms available (except via a user-supplied encoding of implication).

1.5.3 variable abstraction

Can terms be assigned to variables explicitly in formulae or subterms, e.g., using let or where?
let is available, but only "in the large" (i.e. for specification terms).

1.5.4 variable inference

Can/must declarations of variables be left implicit?
Variables must be explicitly bound.
Can variables of unrestricted type ('wild cards') be declared?
No.

1.5.5 lambda-abstraction

Is it allowed in terms?
No.
Notation:

1.5.6 other

Any feature not covered above:

1.6 Other

Any aspect not covered above:

2 Specification in the Large

2.1 Structuring Concepts

2.1.1 structure of specifications v. programs

Does the specification language include a programming language?
No.
If so, what differences are there between specification and program structuring?

2.1.2 structuring logic/institution-independence

Are all the structuring constructs independent of the details of 'specification in the small'?
Yes. Institution independence originated with Clear.

2.1.3 other

Any feature not covered above:

2.2 Basic Constructs

2.2.1 naming

What are the scopes of declared names?
Any specification that includes the specification in which the name is declared.
Notation for referring to particular names:
a **of** *spec-name*; the name *a* alone can be used when it is unambiguous.
How are entities having the same name treated when combined (by union, import, etc.):
They are kept separate unless the names originate in the same specification, in which case they are identified.

2.2.2 union

Notation:
$SP1 + SP2$
Is the combination 'syntactic' or 'semantic':
Semantic: the theory of $SP1 + SP2$ is the closure of the union of the theories of *SP1* and *SP2*.

2.2.3 renaming

Notation:
derive ... **using** ... **from** ... **by** ... **endde**. The signature morphism is in "reverse": *new-name* **is** *old-name*
Are unrenamed entities hidden, preserved, or forbidden?
Hidden.

2.2.4 hiding

Notation:
derive ... **using** ... **from** ... **by** ... **endde**. To get hiding without renaming, the signature should be an inclusion.

2.2.5 sub-specifications

Can parts of specifications be inherited?
Only via enrichment (import).
If so, is multiple inheritance allowed?
Yes, using **enrich** and +.
Notation:
enrich ... **by** ... **enden**

2.2.6 nesting of signatures

Can signatures be nested?
No.
Notation:

2.2.7 morphisms as entities

Notation:
a1 **is** *a2*, ...

2.2.8 behavioural encapsulation

Notation:
Not available.

2.2.9 other basic structuring constructs

Notation and interpretation:
copy ... **using** ...: make a fresh copy of a specification.

2.2.10 other

Any feature not covered above:
Proper treatment of sharing.

2.3 Parametrization

2.3.1 specifications parametrized by programs

Notation:
Not available.

2.3.2 specifications of parametrized programs

Notation:
Not available.

2.3.3 higher-order parametrization

Notation:
Not available.

2.3.4 dependent specifications

Notation:
Not available.

2.3.5 uniqueness of instantiations

Do two identical instantiation expressions denote the same entity?
No.

2.3.6 other

Any feature not covered above:
Ordinary pushout-style parameterised specifications.

2.4 Multiple Logics

2.4.1 heterogeneous specifications

Can several different logics be used in the same specification?
No.

2.4.2 heterogeneous developments

Is it possible to change the logic during development?
No.

2.4.3 logic morphisms, representation morphisms

With heterogeneity, what constructs support the composition/change of logics?

2.4.4 navigation in logic graph

With heterogeneity, what is the relationship between the different logics provided?

2.4.5 composition, instantiation, sub-framework

Notation:

2.4.6 other

Any feature not covered above:

2.5 Other

Any aspect not covered above:
References:

R. Burstall and J. Goguen. Putting theories together to make specifications. *Proc. 5th Intl. Joint Conf. on Artificial Intelligence*, Cambridge, 1045–1058 (1977).

R. Burstall and J. Goguen. The semantics of Clear, a specification language. *Proc. Advanced Course on Abstract Software Specifications*, Copenhagen. Springer LNCS 86, 292–332 (1980).

R. Burstall and J. Goguen. An informal introduction to specifications using Clear. In: *The Correctness Problem in Computer Science* (R. Boyer, J. Moore, eds.). Academic Press 185–213 (1981).

D. Sannella. A set-theoretic semantics for Clear. *Acta Informatica* 21:443–472 (1984).