B

Points to Bear in Mind

B.1 Introduction

- CoFI aims at establishing a wide consensus. 4
- The focus of CoFI is on *algebraic* techniques. 5
- CoFI has already achieved its main aims. 5
- CoFI is an open, voluntary initiative. 6
- CoFI has received funding as an ESPRIT Working Group, and is sponsored by IFIP WG 1.3. 6
- New participants are welcome! 7
- Casl has been designed as a general-purpose algebraic specification language, subsuming many existing languages. 7
- Casl is at the center of a *family* of languages. 8
- Casl itself has several major *parts*. 9

B.2 Underlying Concepts

- Casl is based on standard concepts of algebraic specification. 11
- A basic specification declares symbols, and gives axioms and constraints. 11
- The semantics of a basic specification is a signature and a class of models. 12
- Casl specifications may declare sorts, subsorts, operations, and predicates. 12
- Sorts are interpreted as carrier sets. 12
- Subsorts declarations are interpreted as embeddings. 13
- Operations may be declared as total or partial. 13
- Predicates are different from boolean-valued operations. 13
- Operation symbols and predicate symbols may be overloaded. 14
- Axioms are formulas of first-order logic. 14

M. Bidoit and P.D. Mosses: CASL User Manual, LNCS 2900, pp. 203–210, 2004.

© IFIP International Federation for Information Processing 2004

- Sort generation constraints eliminate 'junk' from specific carrier sets. 15
- The semantics of a structured specification is simply a signature and a class of models. 16
- A translation merely renames symbols. 17
- Hiding symbols removes parts of models. 17
- Union of specifications identifies common symbols. 17
- Extension of specifications identifies common symbols too. 18
- Free specifications restrict models to being free, with initiality as a special case. 18
- Generic specifications have parameters, and have to be instantiated when referenced. 18
- The semantics of an architectural specification reflects its modular structure. 19
- Architectural specifications involve the notions of persistent function and conservative extension. 19
- The semantics of a library of specifications is a mapping from the names of the specifications to their semantics. 20

B.3 Getting Started

- Simple specifications may be written in Casl essentially as in many other algebraic specification languages. 23
- Casl provides also useful abbreviations. 23
- Casl allows loose, generated and free specifications. 24
- Casl syntax for declarations and axioms involves familiar notation, and is mostly self-explanatory. 24
- Specifications can easily be extended by new declarations and axioms. 25
- In simple cases, an operation (or a predicate) symbol may be declared and its intended interpretation defined at the same time. . . . 26
- Symbols may be conveniently displayed as usual mathematical symbols by means of **%display** annotations. 27
- The **%implies** annotation is used to indicate that some axioms are supposedly redundant, being consequences of others. 28
- Attributes may be used to abbreviate axioms for associativity, commutativity, idempotence, and unit properties. 29
- Genericity of specifications can be made explicit using parameters. . . 29
- References to generic specifications always instantiate the parameters. 30
- Datatype declarations may be used to abbreviate declarations of sorts and constructors. 32
- Loose datatype declarations are appropriate when further constructors may be added in extensions. 32
- Sorts may be specified as generated by their constructors. 33
- Generated specifications are in general loose. 34
- Generated specifications need not be loose. 35

- Generated types may need to be declared together. 36
- Free specifications provide initial semantics and avoid the need for explicit negation. 36
- Free datatype declarations are particularly convenient for defining enumerated datatypes. 37
- Free specifications can also be used when the constructors are related by some axioms. 37
- Predicates hold minimally in models of free specifications. 38
- Operations and predicates may be safely defined by induction on the constructors of a free datatype declaration. 38
- More care may be needed when defining operations or predicates on free datatypes when there are axioms relating the constructors. . . 39
- Generic specifications often involve free extensions of (loose) parameters. 40
- Loose extensions of free specifications can avoid overspecification. . . . 41
- Datatypes with observer operations or predicates can be specified as generated instead of free. 42
- The **%def** annotation is useful to indicate that some operations or predicates are uniquely defined. 43
- Operations can be defined by axioms involving observer operations, instead of inductively on constructors. 44
- Sorts declared in free specifications are not necessarily generated by their constructors. 44

B.4 Partial Functions

- Partial functions arise naturally. 47
- Partial functions are declared differently from total functions. 47
- Terms containing partial functions may be undefined, i.e., they may fail to denote any value. 48
- Functions, even total ones, propagate undefinedness. 48
- Predicates do not hold on undefined arguments. 48
- Equations hold when both terms are undefined. 48
- Special care is needed in specifications involving partial functions. . . . 49
- The definedness of a term can be checked or asserted. 50
- The domains of definition of partial functions can be specified exactly. 50
- Loosely specified domains of definition may be useful. 51
- Domains of definition can be specified more or less explicitly. 51
- Partial functions are minimally defined by default in free specifications. 53
- Selectors can be specified concisely in datatype declarations, and are usually partial. 54
- Selectors are usually total when there is only one constructor. 54
- Constructors may be partial. 54

- Existential equality requires the definedness of both terms as well as their equality. 55

B.5 Subsorting

- Subsorts and supersorts are often useful in CASL specifications. 57
- Subsort declarations directly express relationships between carrier sets. 57
- Operations declared on a sort are automatically inherited by its subsorts. 58
- Inheritance applies also for subsorts that are declared afterwards. . . . 59
- Subsort membership can be checked or asserted. 59
- Datatype declarations can involve subsort declarations. 59
- Subsorts may also arise as classifications of previously specified values, and their values can be explicitly defined. 60
- It may be useful to redeclare previously defined operations, using the new subsorts introduced. 61
- A subsort may correspond to the definition domain of a partial function. 62
- Using subsorts may avoid the need for partial functions. 62
- Casting a term from a supersort to a subsort is explicit and the value of the cast may be undefined. 63
- Supersorts may be useful when generalizing previously specified sorts. 64
- Supersorts may also be used for extending the intended values by new values representing errors or exceptions. 65

B.6 Structuring Specifications

- Large and complex specifications are easily built out of simpler ones by means of (a small number of) specification-building operations. . . 67
- Union and extension can be used to structure specifications. 67
- Specifications may combine parts with loose, generated, and free interpretations. 68
- Renaming may be used to avoid unintended name clashes, or to adjust names of sorts and change notations for operations and predicates. 69
- When combining specifications, origins of symbols can be indicated. . 71
- Auxiliary symbols used in structured specifications can be hidden. . . 71
- Auxiliary symbols can be made local when they do not need to be exported. 73
- Care is needed with local sort declarations. 74
- Naming a specification allows its reuse. 75

B.7 Generic Specifications

- Making a specification generic (when appropriate) improves its reusability. 77
- Parameters are arbitrary specifications. 78
- The argument specification of an instantiation must provide symbols corresponding to those required by the parameter. 78
- The argument specification of an instantiation must ensure that the properties required by the parameter hold. 79
- There must be no shared symbols between the argument specification and the body of the instantiated generic specification. 80
- In instantiations, the fitting of parameter symbols to identical argument symbols can be left implicit. 80
- The fitting of parameter sorts to unique argument sorts can also be left implicit. 80
- Fitting of operation and predicate symbols can sometimes be left implicit too, and can imply fitting of sorts. 81
- The intended fitting of the parameter symbols to the argument symbols may have to be specified explicitly. 81
- A generic specification may have more than one parameter. 82
- Instantiation of generic specifications with several parameters is similar to the case of just one parameter. 82
- Composition of generic specifications is expressed using instantiation. 84
- Compound sorts introduced by a generic specification get automatically renamed on instantiation, which avoids name clashes. 85
- Compound symbols can also be used for operations and predicates. 87
- Parameters should be distinguished from references to fixed specifications that are not intended to be instantiated. 88
- Argument specifications are always implicitly regarded as extension of the imports. 89
- Imports are also useful to prevent ill-formed instantiations. 89
- In generic specifications, auxiliary required specifications should be imported rather than extended. 90
- Views are named fitting maps, and can be defined along with specifications. 90
- Views can also be generic. 91

B.8 Specifying the Architecture of Implementations

- Architectural specifications impose structure on implementations, whereas specification-building operations only structure the text of specifications. 93

- An architectural specification consists of a list of unit declarations, specifying the required components, and a result part, indicating how they are to be combined. 96
- There can be several distinct architectural choices for the same requirements specification. 97
- Each unit declaration listed in an architectural specification corresponds to a separate implementation task. 97
- A unit can be implemented only if its specification is a conservative extension of the specifications of its given units. 98
- Genericity of components can be made explicit in architectural specifications. 100
- A generic component may be applied to an argument richer than required by its specification. 101
- Specifications of components can be named for further reuse. 102
- Both named and unnamed specifications can be used to specify components. 102
- Specifications of generic components should not be confused with generic specifications. 103
- A generic component may be applied more than once in the same architectural specification. 103
- Several applications of the same generic component is different from applications of several generic components with similar specifications. 104
- Generic components may have more than one argument. 105
- Open systems can be described by architectural specifications using generic unit expressions in the result part. 106
- When components are to be combined, it is best to check that any shared symbol originates from the same non-generic component. 107
- Auxiliary unit definitions or local unit definitions may be used to avoid repetition of generic unit applications. 109

B.9 Libraries

- Libraries are named collections of named specifications. 111
- Local libraries are self-contained. 111
- Distributed libraries support reuse. 111
- Different versions of the same library are distinguished by hierarchical version numbers. 112
- Local libraries are self-contained collections of specifications. 112
- Specifications can refer to previous items in the same library. 113
- All kinds of named specifications can be included in libraries. 114
- Display, parsing, and literal syntax annotations apply to entire libraries. 114
- Libraries and library items can have author and date annotations. 116
- Libraries can be installed on the Internet for remote access. 116

- Validated libraries can be registered for public access. 117
- Libraries should include appropriate annotations. 118
- Libraries can include items downloaded from other libraries. 118
- Substantial libraries of basic datatypes are already available. 119
- Libraries need not be registered for public access. 120
- Subsequent versions of a library are distinguished by explicit version numbers. 120
- Libraries can refer to specific versions of other libraries. 121
- All downloadings should be collected at the beginning of a library. 122

B.10 Foundations

- A complete presentation of Casl is in the *Reference Manual*. 125
- Casl has a definitive summary. 125
- Casl has a complete formal definition. 126
- Abstract and concrete syntax of Casl are defined formally. 126
- Casl has a complete formal semantics. 126
- Casl specifications denote classes of models. 127
- The semantics is largely institution-independent. 127
- The semantics is the ultimate reference for the meanings of all Casl constructs. 128
- Proof systems for various layers of Casl are provided. 128
- The foundations of our Casl are rock-solid! 129

B.11 Tools

- Casl specifications can be checked for well-formedness using a form-based web page. 131
- The Heterogeneous Tool Set (Hets) is the main analysis tool for Casl. 132
- Hets can be used for parsing and checking static well-formedness of specifications. 133
- Hets also displays and manages proof obligations, using development graphs. 134
- Nodes in a development graph correspond to Casl specifications. Arrows show how specifications are related by the structuring constructs. 135
- Internal nodes in a development graph correspond to unnamed parts of a structured specification. 137
- Hol-Casl is an interactive theorem prover for Casl, based on the tactical theorem prover Isabelle. 138
- Casl is linked to Isabelle/Hol by an encoding. 138
- Asf+Sdf was used to prototype the Casl syntax. 139

- The Asf+Sdf Meta-Environment provides syntax-directed editing of Casl specifications. 140

B.12 Basic Libraries

- The Casl Basic Libraries contain the standard datatypes. 143
- Hets can be used to get an overview of the Basic Libraries. 143