CASL

Common Algebraic Specification Language

Overview

Peter D. Mosses (BRICS, Aarhus, Denmark)

External Relations Coordinator of

CoFI: The Common Framework Initiative

for algebraic specification and development

Plan

- Background
- CASL
  - basic specifications
  - structured specifications
  - architectural specifications
  - libraries of specifications
- Foreground

Background

- Lack of a common framework meant
  - industrial acceptance: *discouraged*
  - dissemination: *hindered*
  - tool applicability: *limited*
- Why not agree on a common framework?
  - concepts
  - language
  - methodologies

Aims

- *a common framework*—a collaborative effort
- for *algebraic* specification & development
- *attractive*
  - short term: to researchers; long term: for use in industry
- providing a *common specification language*
  - syntax: uniform, user-friendly; semantics: straightforward
- subsuming *many* previous frameworks
- good documentation & *tool support*
- *free*—but protected!

Scope

- specifying functional requirements
- formal development & verification of software
- relation of specifications to:
  - informal requirements
  - implemented code
- prototyping, theorem-proving
- libraries, reuse, evolution
- providing tool interoperability

CASL:
Common Algebraic Specification Language

- critical selection of existing constructs
- expressive, simple, pragmatic
- specifying requirements and design
  for conventional software packages
- restrictions to sub-languages
- extensions to higher-order, state-based, concurrent, …

CASL Design Process

- design started: September 1995
- design proposed: May 1997
  - summary, abstract syntax, semantics
  - tentatively approved by IFIP WG1.3
- design v1.0 finalized: October 1998
  - language documentation available
    - » summary, formal semantics, rationale, design notes
    - » user's manual in preparation (draft by Sept. 1999)
  - tools are being developed (parsers, provers, …)

Parts of CASL

- Basic specifications
  - declarations, definitions, axioms
- Structured specifications
  - translations, reductions, unions, extensions, freeness
  - named specifications, generic specifications, views
- Architectural specifications
  - implementation units, composition
- Specification libraries
  - distributed

Aspects of CASL design

- Design issues
  - methodology, tools, aesthetics
- Semantic concepts
  - formal (institutions, environments)
  - informal (expansions, scopes)
- Language constructs
  - abstract syntax (structure, annotations)
  - concrete syntax (input, display)

Design issues: basic specifications

- Partial and total functions
- Logic and predicates
- Classes of models
- Overloading
- Subsorts
- Datatype constructors/selectors

Partial and total functions

- error propagation implicit (strictness)
  - any argument undefined $\Rightarrow$ result undefined
- totality an important special case
  - all arguments defined $\Rightarrow$ result defined
- domain of definition may be implicit
  - not always a natural sort/subsort
- equations may require definedness or not
  - existential equations require it, strong do not

Logic and predicates

- classical 2-valued first-order logic
  - standard interpretation of connectives
- predicates
  - better than (total) Boolean functions!
  - any argument undefined $\Rightarrow$ unsatisfied
- quantification, connectives
  - expressiveness
  - restrictions in sublanguages

Classes of models

- all models
  - loose semantics
- generated models
  - only expressible values considered
  - induction proofs sound
- initial (and freely-generated) models
  - minimal satisfaction, definedness
  - may not exist (disjunction, negation, ...)

Overloading

- declaration
  - same symbol, various profiles (arg/result sorts)
    - » e.g. ‘+’ on integers, reals, strings, …
- use
  - intended profile determined by context
  - disambiguation when needed by specifying profile (or result sort) of applications

Subsorts

- embeddings
  - not necessarily inclusions (e.g. Int values represented differently from Real values)
- commute with overloaded functions
  - values independent of profiles
  - 2+2=4 (natural numbers, integers, …)
- no ‘regularity’ conditions on declarations
  - for simplicity, modularity

Datatype constructors/selectors

- frequently needed
  - correspond to (unions of) record types, enumeration types
- abbreviate tedious declarations, axioms
  - for constructors and selectors
- may be loose, generated, or free

Semantic concepts: basic specifications

- Signatures
- Models
- Sentences
- Satisfaction
- Sort generation constraints
- Institution
- Semantic functions

Basic specification example

free types *Nat* ::= *0* | **sort** *Pos*;
Pos ::= *suc*(*pre*:*Nat*)
op *pre* : *Nat* →? *Nat*
axioms ¬ *def* *pre*(*0*);
∀*n*:*Nat* • *pre*(*suc*(*n*)) = *n*

pred *even*__ : *Nat*
var *n*:*Nat*
• *even 0*
• *even suc*(*n*) ⇔ ¬ *even n*

Design issues: structured specifications

- No model structure
- Names of symbols
- Generic extension

No model structure

- structure of specification
  ≠ specification of structure!
- models of structured specifications
  – same kind as for basic specifications
- separate architectural specifications

Names of symbols

- general principle:
  same name, same thing!
  - the “name” of an operation or predicate includes its profile, so overloading does not require identification (except with subsorts)
- applies both to unions and extensions
  - but not to named specifications in libraries
  - clashes can be eliminated by renaming

Names of symbols…

- auxiliary symbols are to be hidden
  - hidden symbols are irrelevant
  - revealed symbols are significant
- tools may warn about duplicate declarations
  - no problem when same origin

Generic extension

- specification definitions
  - reuse by reference to name
- generic parameters
  - parameters intended to vary
  - reuse requires instantiation
  - not intended for defining arbitrary functions on specifications (macros)

Semantic concepts: structured specifications

- signature, class of models
  - as for basic specifications
- institution-independent structure
  - many-sorted, subsorted, …
  - reduct homomorphisms
- local environment
  - visible symbol declarations

Structured specification example

```
spec LIST_WITH_ORDER [PARTIAL_ORDER] =
     free type List[Elem] ::= nil | cons(hd:?Elem; tl:?List[Elem])
then local op    insert : Elem × List[Elem] → List[Elem];
           vars  x,y : Elem; l : List[Elem]
           axioms
           insert(x,nil) = cons(x,nil);
           x≤y ⇒ insert(x,cons(y,l)) = cons(x,insert(y,l));
           ¬(x≤y) ⇒ insert(x,cons(y,l)) = cons(y,insert(x,l))
within     op    order[__≤__] : List[Elem] → List[Elem]
           vars  x : Elem; l : List[Elem]
           axioms
           order[__≤__](nil) = nil;
           order[__≤__](cons(x,l)) = insert(x,order[__≤__](l))
end
```

Design issues: architectural specifications

- Reusability
- Interfaces
- Decomposition of tasks – composition of results

Reusability

- structure of specifications $\Rightarrow$ reusable specifications
- specification of structure $\Rightarrow$ reusable design/implementation units
  - e.g. structure implementation of LIST [NAT] as:
    - » implementation N of NAT and
    - » a function F extending any such N to an implementation of LIST [NAT] and
    - » compose by applying F to N

Interfaces

- explicit assumptions that parts of the implementation make about others
- explicit guarantees about results under the given assumptions

Decomposition/composition

- architectural specifications specify decomposition of implementation development task into sub-tasks
- also how to compose results of sub-tasks into result of main task

Semantic concepts: architectural specifications

- model of architectural specification:
  - collection of (named) unit constructors
  - unit (constructor) resulting from composition
- unit constructors:
  - constants
  - functions from units to units
    - » persistent, extending arguments
    - » arguments must be compatible

Architectural specification example

spec LIST[ELEM] =
  free type *List[Elem] ::=*
    nil | *cons(Elem; List[Elem])*

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

Design issues: libraries

- visibility of symbols between specifications
  - none
- visibility of specification names
  - linear
- distributed libraries on Internet
  - indirect links
  - explicit downloading
- version control

Semantic concepts: libraries

- global environment
  - mapping specification names to their denotations
- directory of registered libraries
  - mapping library names to their registered URLs
  - giving access to previous versions of libraries
- state of the universe (Internet, local files)

Foreground: CoFI Working Group

- Language design tasks
- Semantics tasks
- Tools tasks
- Methodology tasks
- Reactive system tasks
- Overall coordination:
  - Don Sannella (Edinburgh) Sep 98 – …
  - Peter D. Mosses (Aarhus) Sep 95 – Aug 98

Language design tasks

- coordination:
  - Bernd Krieg-Brückner (Bremen)
- publication of CASL Language Summary
- sub-languages
  - total, many-sorted, equational, …
- extensions
  - higher-order, state-based, …

Semantics tasks

- coordination:
  - Andrzej Tarlecki (Warsaw)
- publication of formal semantics for CASL
- choice of proof system for CASL

Methodology tasks

- coordination:
  - Michel Bidoit (Cachan)
- user's guide
- case studies
- relation of specifications to requirements and code
- development processes

Tools tasks

- coordination:
  - Hélène Kirchner (Nancy)
- parsers
  - 3 prototypes implemented
- interoperability of existing tools
- formatting (LaTeX package, HTML filter)
- static analysis, proof tools
  - HOL/Isabelle interface for CASL implemented

Reactive system tasks

- coordination:
  - Egidio Astesiano (Genova)
- extensions of CASL for reactive system specification
- combination of CASL with concurrency

External relations tasks

- coordination:
  - Peter D. Mosses (Aarhus)
- relation to existing languages
  - ASF+SDF, CAFEOBJ, LSL (so far superficially)
  - help needed from experts!
- liaison with IFIP WG1.3
- web presentation of CoFI
- tutorial

Join now!

- all task groups welcome new participants
  - contact the coordinators
- all cofi-$*$ mailing lists open to subscription
  - moderated
  - administered by Majordomo program
  - cofi-list@brics.dk:
    low-volume, announcements only
- all documents available on the web/ftp

References

- CoFI Home Page
  - http://www.brics.dk/Projects/CoFI/
- CASL Summary
  - .../Documents/CASL/Summary/
- CoFI Rationale, CASL Guided Tour
  - *EATCS Bulletin* 59, June1996
  - Proc. TAPSOFT'97, *LNCS* 1214, 1997
  - Proc. WADT'98, *LNCS* 1589, 1999