Software specification in CASL - The Common Algebraic Specification Language

Till Mossakowski

Summer 2014

Overview

- Why formal specification?
- Waterfall Model
- Example: sorting
- CASL – the Common Algebraic Specification Language
- Layers of CASL
- Overview of the course

Why formal specification?

Erroneous software systems may lead to

- *economic losses*
  (e.g.: loss of Ariane V and mars probe, pentium bug),
- *security problems* (e.g.: viruses),
- *damage of persons* (e.g.: death due to erroneously computed radiation dose)

Why formal specification?

Erroneous software systems may lead to

- *economic losses*
  (e.g.: loss of Ariane V and mars probe, pentium bug),
- *security problems* (e.g.: viruses),
- *damage of persons* (e.g.: death due to erroneously computed radiation dose)

Why formal specification?

Erroneous software systems may lead to

- *economic losses*
  (e.g.: loss of Ariane V and mars probe, pentium bug),
- *security problems* (e.g.: viruses),
- *damage of persons* (e.g.: death due to erroneously computed radiation dose)

Formal specification — Success stories

- complete formal verification of *microprocessor arithmetic* (pentium 4, AMD)
- NASA uses axiomatic specification of *physical units*
- verification of the *Java bytecode verifier*
- found 12 deadlocks in Occam code for *international space station*

Formal specification — Success stories

- complete formal verification of *microprocessor arithmetic* (pentium 4, AMD)
- NASA uses axiomatic specification of *physical units*
- verification of the *Java bytecode verifier*
- found 12 deadlocks in Occam code for *international space station*

Formal specification — Success stories

- complete formal verification of *microprocessor arithmetic* (pentium 4, AMD)
- NASA uses axiomatic specification of *physical units*
- verification of the *Java bytecode verifier*
- found 12 deadlocks in Occam code for *international space station*

Formal specification — Success stories

- complete formal verification of *microprocessor arithmetic* (pentium 4, AMD)
- NASA uses axiomatic specification of *physical units*
- verification of the *Java bytecode verifier*
- found 12 deadlocks in Occam code for *international space station*

Axiomatic Specfication

- *loose requirements*, close to informal descriptions
- *clarification* of underlying mathematical *concepts*
- *design* of algorithms and data structures *independently* of any implementation language
- CASL is a *standard* for axiomatic specification

- *loose requirements*, close to informal descriptions
- *clarification* of underlying mathematical *concepts*
- *design* of algorithms and data structures *independently* of any implementation language
- CASL is a *standard* for axiomatic specification

Axiomatic Specfication

- *loose requirements*, close to informal descriptions
- *clarification* of underlying mathematical *concepts*
- *design* of algorithms and data structures *independently* of any implementation language
- CASL is a *standard* for axiomatic specification

Axiomatic Specfication

- *loose requirements*, close to informal descriptions
- *clarification* of underlying mathematical *concepts*
- *design* of algorithms and data structures *independently* of any implementation language
- CASL is a *standard* for axiomatic specification

Waterfall Model (slide by M. Roggenbach)

	Requirement Elicitation and Analysis	
	↑↓	
Nat Lang.	Informal Specification	
	↑↓	Validation
Spec. Lang.	Formal Requirements Specification	
	↑↓	"Invent & Verify",
Spec. Lang.	Formal Design Specification	
	↑↓	Transformation or
Progr. Lang.	Implementation	
	↑↓	Systematic Testing
	Test	
	↑↓	
	Maintenance	

Example: sorting

Informal specification:

To sort a list means to find a list with the same elements, which is in ascending order.

Formal *requirements* specification:

- $is\_ordered(sorter(L))$
- $is\_ordered(L) \Leftrightarrow \forall L1, L2 : List;\; x, y : Elem\,.$
  $$L = L1 ++ [x, y] ++ L2 \Rightarrow x \leq y$$
- $permutation(L, sorter(L))$
- $permutation(L1, L2) \Leftrightarrow$
  $$\forall x : Elem\,.\; count(x, L1) = count(x, L2)$$

Example: sorting

Informal specification:

To sort a list means to find a list with the same elements, which is in ascending order.

Formal *requirements* specification:

- $is\_ordered(sorter(L))$
- $is\_ordered(L) \Leftrightarrow \forall L1, L2 : List;\ x, y : Elem\ .$
  $$L = L1 ++ [x, y] ++ L2 \Rightarrow x \leq y$$
- $permutation(L, sorter(L))$
- $permutation(L1, L2) \Leftrightarrow$
  $$\forall x : Elem\ .\ count(x, L1) = count(x, L2)$$

Sorting (cont'd)

We want to show insert sort to enjoy these properties.
Formal *design specification*:

- $insert(x, []) \ = \ [x]$
- $insert(x, y :: L) \ =$
  $x :: y :: L) \ \ when \ \ x \leq y$
  $else \ \ y :: insert(x, L)$
- $insert\_sort([]) \ = \ []$
- $insert\_sort(x :: L) \ = \ insert(x, insert\_sort(L))$

Implementation (in Haskell)

```
insert :: Ord a => (a,[a]) -> [a]
insert(x,[]) = [x]
insert(x,y:l) = if x <= y then x:y:l
                   else y:insert(x,l)

insert_sort :: Ord a => [a] -> [a]
insert_sort([]) = []
insert_sort(x:l) = insert(x,insert_sort(l))
```

- de facto *standard* for specification of functional requirements
- developed by the "Common Framework Initiative" (an *open* international collaboration)
- approved by *IFIP WG 1.3* "Foundations of Systems Specifications"
- CASL *User Manual* (Lecture Notes in Computer Science 2900) and *Reference Manual* (Lecture Notes in Computer Science 2960)

- de facto *standard* for specification of functional requirements
- developed by the “Common Framework Initiative”
  (an *open* international collaboration)
- approved by *IFIP WG 1.3*
  “Foundations of Systems Specifications”
- CASL *User Manual* (Lecture Notes in Computer Science 2900) and *Reference Manual* (Lecture Notes in Computer Science 2960)

- de facto *standard* for specification of functional requirements
- developed by the "Common Framework Initiative"
  (an *open* international collaboration)
- approved by *IFIP WG 1.3*
  "Foundations of Systems Specifications"
- CASL *User Manual* (Lecture Notes in Computer Science 2900) and *Reference Manual* (Lecture Notes in Computer Science 2960)

CASL – the Common Algebraic Specification Language

- de facto *standard* for specification of functional requirements
- developed by the "Common Framework Initiative" (an *open* international collaboration)
- approved by *IFIP WG 1.3* "Foundations of Systems Specifications"
- CASL *User Manual* (Lecture Notes in Computer Science 2900) and *Reference Manual* (Lecture Notes in Computer Science 2960)

Foundations of CASL

- detailed *language summary*, with informal explantation
- formal definition of *abstract and concrete syntax*
- complete *formal semantics*
- *proof systems*
- libraries of *basic datatypes*

All this is contained in the *Reference Manual*
— here, we will largely follow the *User Manual*

- detailed *language summary*, with informal explantation
- formal definition of *abstract and concrete syntax*
- complete *formal semantics*
- *proof systems*
- libraries of *basic datatypes*

All this is contained in the *Reference Manual*
— here, we will largely follow the *User Manual*

Foundations of CASL

- detailed *language summary*, with informal explantation
- formal definition of *abstract and concrete syntax*
- complete *formal semantics*
- *proof systems*
- libraries of *basic datatypes*

All this is contained in the *Reference Manual*
— here, we will largely follow the *User Manual*

Foundations of CASL

- detailed *language summary*, with informal explantation
- formal definition of *abstract and concrete syntax*
- complete *formal semantics*
- *proof systems*
- libraries of *basic datatypes*

All this is contained in the *Reference Manual*
— here, we will largely follow the *User Manual*

Foundations of CASL

- detailed *language summary*, with informal explantation
- formal definition of *abstract and concrete syntax*
- complete *formal semantics*
- *proof systems*
- libraries of *basic datatypes*

All this is contained in the *Reference Manual*
— here, we will largely follow the *User Manual*

Foundations of CASL

- detailed *language summary*, with informal explantation
- formal definition of *abstract and concrete syntax*
- complete *formal semantics*
- *proof systems*
- libraries of *basic datatypes*

All this is contained in the *Reference Manual*
— here, we will largely follow the *User Manual*

CASL has rock-solid foundations

- the complete formal semantics maps the syntax to underlying mathematical concepts
- CASL specifications denote classes of models
- The semantics is largely indepdendent of the details of the logic (institution)
- The semantics is the ultimative reference for the meaning of CASL

CASL has rock-solid foundations

- the complete formal semantics maps the syntax to underlying mathematical concepts
- CASL specifications denote classes of models
- The semantics is largely indepdendent of the details of the logic (institution)
- The semantics is the ultimative reference for the meaning of CASL

CASL has rock-solid foundations

- the complete formal semantics maps the syntax to underlying mathematical concepts
- CASL specifications denote classes of models
- The semantics is largely indepdendent of the details of the logic (institution)
- The semantics is the ultimative reference for the meaning of CASL

CASL has rock-solid foundations

- the complete formal semantics maps the syntax to underlying mathematical concepts
- CASL specifications denote classes of models
- The semantics is largely indepdendent of the details of the logic (institution)
- The semantics is the ultimative reference for the meaning of CASL

CASL on the web

- CASL in general: http://www.cofi.info
- CASL tools: http://hets.dfki.de
- CASL libraries: http://www.cofi.info/Libraries

Layers of CASL

CASL consists of several major *layers*, which are quite independent and may be understood (and used) separately:

Basic specifications many-sorted first-order logic, subsorting, partial functions, induction, datatypes.

Structured specifications translation, reduction, union, and extension of specifications; generic (parametrized) and named specifications

Layers of CASL

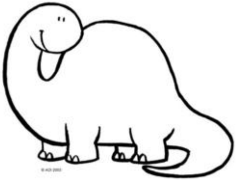
CASL consists of several major *layers*, which are quite independent and may be understood (and used) separately:

Basic specifications many-sorted first-order logic, subsorting, partial functions, induction, datatypes.

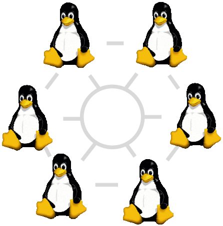
Structured specifications translation, reduction, union, and extension of specifications; generic (parametrized) and named specifications

Why Modular Decomposition?

- reduction of *complexity*
- better *understanding* of specification and code (small pieces, well-defined interfaces)
- better *distribution of work*



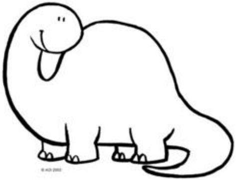
vs.



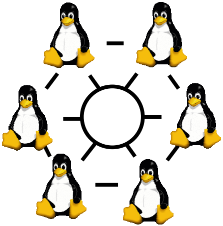
- better *maintenance* and possibilities of *re-use*

Why Modular Decomposition?

- reduction of *complexity*
- better *understanding* of specification and code (small pieces, well-defined interfaces)
- better *distribution of work*



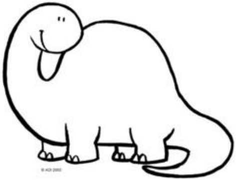
vs.



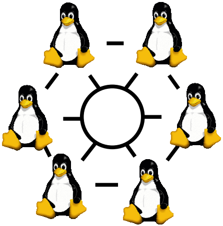
- better *maintenance* and possibilities of *re-use*

Why Modular Decomposition?

- reduction of *complexity*
- better *understanding* of specification and code (small pieces, well-defined interfaces)
- better *distribution of work*



vs.



- better *maintenance* and possibilities of *re-use*

Layers of CASL (cont'd)

Architectural specifications structuring of *implementation*: define how models of a specification may be constructed out of models of simpler specifications.

Libraries allow the distributed (over the Internet) storage and retrieval of (particular versions of) named specifications.

Layers of CASL (cont'd)

Architectural specifications structuring of *implementation*: define how models of a specification may be constructed out of models of simpler specifications.

Libraries allow the distributed (over the Internet) storage and retrieval of (particular versions of) named specifications.

Overview of the course

- recall basics of first-order logic
- loose + free specifications (case study: text formatting)
- CASL tools: Hets and SPASS
- partial functions, subsorting
- generated specifications
- a bit of semantics
- structuring and generic specifications
- architectural specifications
- case studies (invoice system, steam boiler, ontologies)
- outlook: CASL extensions

Continue with slides for CASL User Manual (by M. Bidoit and P.D. Mosses)